Zach Siegel (#805435913), Edward Nguyen (#904663048), Boya (#005037574), Banseok Lee (#605351891)

EE219 - *Large-Scale Data Mining: Models and Algorithms*

Prof. Vwani Roychowdhury

Winter 2021 - Project 2

Question 1.

There are 4732 samples and a vocabulary of size 15547, so the TF-IDF matrix has shape 4732×15547, as can be seen in Figure 1.

```
TF-IDF Matrix Shape (all data): (4732, 15547)
```

Figure 1: Console output for the dimensions of the TF-IDF training and test matrices.

For this project, we excluded English stopwords, excluded words that appear in fewer than 3 documents in the corpus (`min_df=3`), did not stem or lemmatize, and removed both the headers and the footers of each article.

Question 2.

Figure 2 shows the contingency table for clustering with $k = 2$ clusters and 2 true classes.

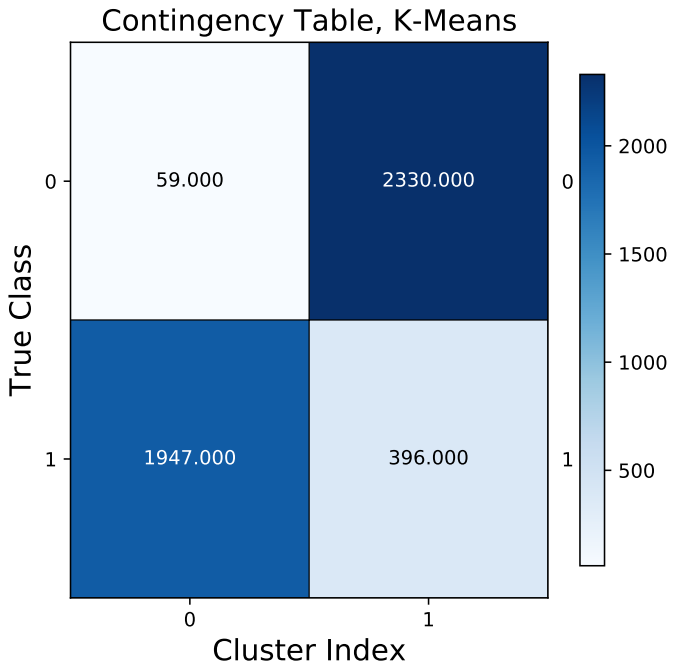


Figure 2: Contingency table of K-Means clustering with $k = 2$ clusters and 2 true classes.

Calculating a contingency table is straightforward in Python:

```
def get_contingency(targets,predictions):
  A = np.zeros((len(set(targets)),len(set(predictions))))
  for (i,j) in zip(targets,predictions):
    A[i,j] += 1
  return A
```

or in one line:

```
def get_contingency(targets,predictions):
    return np.array([[sum([1 if targets[i]==c and predictions[i]==k else 0 for i in range(len(targets))]) for k in
        range(len(set(predictions)))] for c in range(len(set(targets)))])
```

Question 3.

We report the homogeneity score, completeness score, v-measure, adjusted rand index, and adjusted mutual information score in Figure 3. We also report one additional score ("balanced accuracy score").

```
{'adjusted_rand_score': 0.652294316321755,
 'balanced_accuracy_score': 0.09685530512501547,
 'completeness_score': 0.5841527500588086,
 'homogeneity_score': 0.5743984789472549,
 'mutual_info_score': 0.3981155457956391,
 'v_measure_score': 0.5792345520957184}
```

Figure 3: Performance metrics for K-means clustering with $k = 2$ clusters and 2 true classes.

The `sklearn` package cites [4] for their definitions of homogeneity and completeness, which are defined in terms of the "entropy" and "conditional entropy" of the class labels and the cluster labels. Suppose there are N total observations, K clusters, C classes, and a_{ck} observations in class c assigned to cluster k. The entropies are defined in [4] as

$$
\begin{aligned}
H(C|K) &= -\sum_{k=1}^{K}\sum_{c=1}^{C}\frac{a_{ck}}{N}\log\left(\frac{a_{ck}}{\sum_{c=1}^{C}a_{ck}}\right),\\
H(K|C) &= -\sum_{c=1}^{C}\sum_{k=1}^{K}\frac{a_{ck}}{N}\log\left(\frac{a_{ck}}{\sum_{k=1}^{K}a_{ck}}\right),\\
H(C) &= -\sum_{c=1}^{C}\frac{\sum_{k=1}^{K}a_{ck}}{C}\log\left(\frac{\sum_{k=1}^{K}a_{ck}}{C}\right)\\
H(K) &= -\sum_{k=1}^{K}\frac{\sum_{c=1}^{C}a_{ck}}{C}\log\left(\frac{\sum_{c=1}^{C}a_{ck}}{C}\right).
\end{aligned}
$$

and the homogeneity, completeness, and v-measure are given as[1]

- Homogeneity h is calculated as

$$\text{homogeneity} = 1 - \frac{H(C|K)}{H(C)},$$

- Completeness c is calculated as

$$\text{completeness} = 1 - \frac{H(K|C)}{H(K)}$$

[1] I don't accept this definition at face value, as the definition of $H(C)$ and $H(K)$ are always negative, leading to homogeneity and completeness > 1. In the `sklearn` source code, rather than directly using entropy, the mutual information score is used. Since the `sklearn` documentation cites [4] and the source code calculates homogeneity and completeness differently, I am comfortable assuming that paper may have a calculation error.

- The v-measure is calculated as follows:

$$v = \frac{(1+\beta) \times \text{homogeneity} \times \text{completeness}}{\beta \times \text{homogeneity} + \text{completeness}}$$

  and `sklearn` defaults to $\beta = 1$.

Question 4.

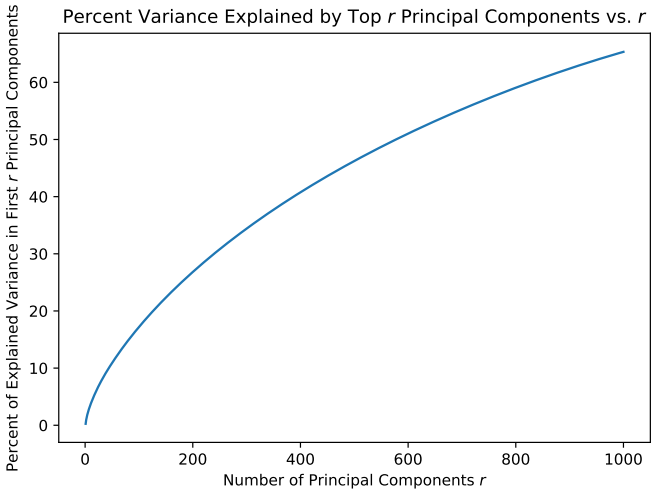


Figure 4: Percent variance in tf-idf data captured by the first r principal components, $r \in \{1, \ldots, 1000\}$

The amount of variance captured by each successive principal component is less than or equal to that of the previous. In fact, by definition, each principal component captures a maximal amount of the variance of the residual between the data and its projection onto the previous principal components. Still, it is not guaranteed that the variance captured by subsequent principal components will be strictly less than that captured by the previous ones - for example, if data are balanced linear combinations of already-orthogonal bases.

In practice, often vectorized "natural" data has a lower "implicit" dimension that that of its representation, and the majority of its variation is captured in far fewer principal components than that dimension. "Natural" data can include image, video, recorded audio, text, and even behavioral data. In this case, each article is represented in the TF-IDF matrix as a vector in $\mathbb{R}^{15547}$, reflecting a vocabulary of over $15,000$ words, yet over 60% of the variance in the data is captured in just the first 1000 principal components, indicating an "implicit vocabulary" of only around 1000 elements that can describe the articles fairly well.

Question 5.

For $r \in \{1, 2, 3, 5, 10, 20, 50, 100, 300\}$, we represent each article in $\mathbb{R}^r$ prior to applying K-means clustering and calculating each of 6 purity metrics. For each r, we reduce dimension in the following two ways:

- We project each article onto the first r principal components, and
- We calculate a non-negative matrix factorization with r bases.

The purity metrics over this range of r-values is visualized in Figure 5.

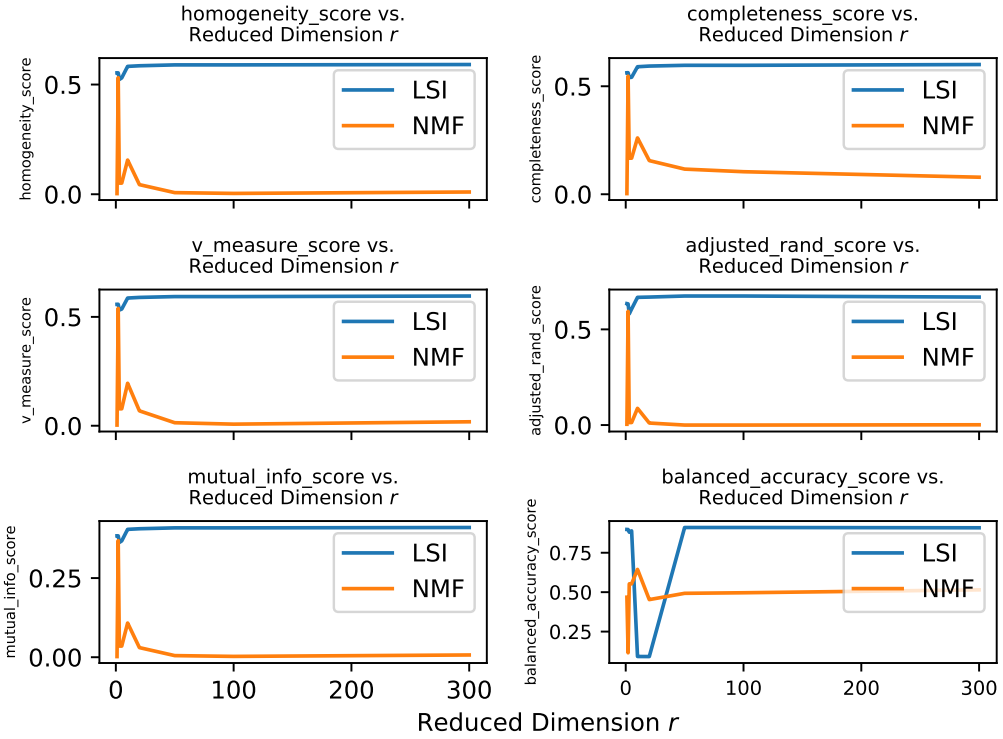


Figure 5: Six measures of clustering "purity" (one extra measure to make an even number) versus dimension of data before K-Means clustering.

For all metrics at almost all dimensions r, dimensionality reduction using LSI seems to yield better clustering purity than NMF. Interestingly, LSI seems to perform reasonably with $r = 1$ dimensions, whereas in almost all cases, NMF performs very poorly in that case. LSI suffers from adding dimensions when only a small number are used, but then almost monotonically improves, though just barely. NMF seems to perform very well with $r = 2$ and $r = 10$ dimensions with several of the purity metrics but poorly with most other dimensions.

A "good" choice of r for the two metrics is somewhat subjective, but

- For LSI, these metrics imply that a "good" choice of r is $r = 300$. LSI seems to improve slightly with additional dimensions up to and including $r = 300$. That being said, LSI improves *very little*

from $r = 10$ to $r = 300$, so, despite better clustering purity metrics at $r = 300$, it would also be reasonable to say that $r = 10$ is a "good" choice for LSI.

- For NMF, a "good" choice of r seems to be $r = 10$.

Question 6.

As r increases, the L_2-norm used in the K-means objective function does a poorer and poorer job of capturing proximity of observations. Indeed, "most of the area of a high-dimensional orange lies in the skin" describes this phenomenon, and shows that in high dimensions, Euclidean distance does not capture relative proximity in quite the same way it does in lower dimensions. Furthermore, higher dimensions produce harder optimization problems, though we ensured that there were sufficient iterations for the `sklearn` K-means implementation to converge in our results. For primarily the former reason (distances poorly capturing relative proximity in high dimensions), a higher-dimensional representation of data yields poor clustering performance in general. This motivates the dimensionality-reduction of LSI or NMF.

On the other hand, data is represented more accurately with more dimensions. Figure 4 demonstrates that 1000-dimensional LSI representation of the TF-IDF data (in $\sim \mathbb{R}^{15000}$ before dimensionality reduction) captures only roughly 60% of the variation in the observations. An NMF representation is more constrained than an LSI representation (the matrix factorization is constrained to be non-negative as well as accurate), implying an *even less accurate* low-dimensional representation of the data via NMF. Some of the features of the TF-IDF data that might differentiate articles into clusters corresponding to their class may not be accurately represented in low dimensions. For this reason, cluster purity may improve as the data is represented with higher dimensions r.

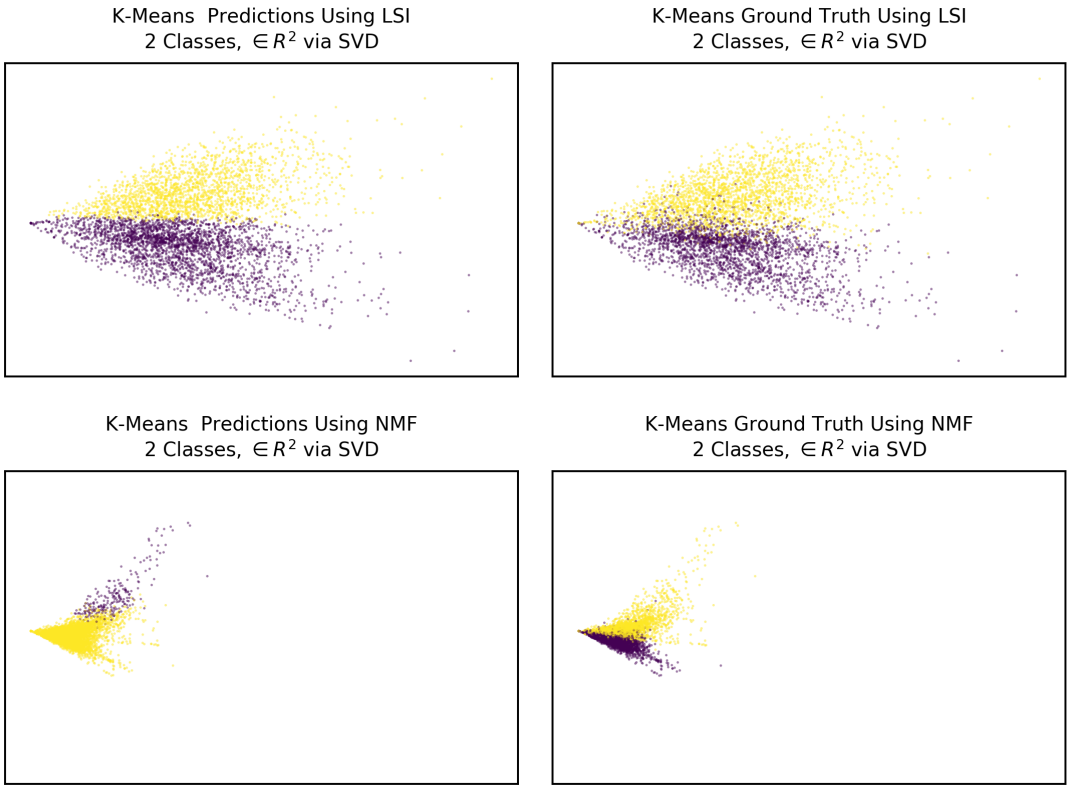
Due to these conflicting and very different phenomena - K-means performance versus data representation accuracy - there is no reason for cluster purity performance to strictly increase or decrease as the dimensionality of the data representation r increases.

Note that recent successes in machine learning, such as those in facial recognition, have also involved both a *dimensionality-reducing* and a *learning* stage, but with an important difference. LSI reduces dimensionality in a way that is *as accurate as possible* with respect to L_2 loss, but may still compromise aspects of the data that prove useful for later classification tasks. The recently-successful alternative to this is a task-based dimensionality-reduction scheme, in which dimensionality-reduction is treated as a differentiable layer in a classification network, whose parameters are trained via backpropagation in stochastic gradient descent, along with the parameters of the learning (classification or clustering) layers. This can be as complicated as a fully-connected neural network with a "narrow" layer producing dimension-reduced representations before classification; or as simple as a weighted matrix factorization, with respect to whose weights the factorization is differentiable, preceding logistic regression (whose parameters are also differentiable). Numerical differentiation frameworks, like those underlying Tensorflow or Pytorch, are eminently capable of implementing such a scheme. In that case, the low-dimensional representations may be "inaccurate" with respect to L_2 loss (which, as discussed above, is shown to pose difficulties in all but very-low dimensions), but result in very accurate predictions or very pure clusters (if a differentiable clustering method is used). For an example of differentiable dimensionality-reduction and differentiable clustering, see [1]; the concept of differentiable dimensionality-reduction (or differentiable representation in general), supplanting or complementing traditional loss-minimizing rep-

resentations (such as PCA/SVD/LSI and NMF), has played a key role in many of the recent successes of deep learning.

Question 7.

We choose to represent data in $\mathbb{R}^{20}$ for both LSI and NMF before clustering. This is an attempt to demonstrate an "apples to apples" comparison of LSI and NMF for this particular task.




```
4732 Samples, 2 Classes, 2 Clusters

Clustering Performance After LSI Dimensionality Reduction:
{'adjusted_rand_score': 0.6612019530507731,
 'balanced_accuracy_score': 0.9059558972363553,
 'completeness_score': 0.586364214905493,
 'homogeneity_score': 0.578198554529493,
 'mutual_info_score': 0.40074937791730514,
 'v_measure_score': 0.582252756801881}

Clustering Performance After NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.0105203305797331,
 'balanced_accuracy_score': 0.4528176964165857,
 'completeness_score': 0.15572667633730974,
 'homogeneity_score': 0.04381643723458578,
 'mutual_info_score': 0.030369169598844475,
 'v_measure_score': 0.0683901139692893}
```

Figure 6: Points correspond to news articles of 2 classes ("Recreational Sports" and "Computer Technology") in tf-idf representation reduced to $\mathbb{R}^{20}$ by LSI (top row) or NMF (bottom row), clustered into $k = 2$ clusters using K-Means, then reduced to $\mathbb{R}^2$ by LSI (PCA/SVD). In the left column, points are colored by their assigned cluster; clearly, K-Means assigns the same cluster to points that are close together as a rule. In the right column, points are colored by their true class; clearly, points (articles) of the same class are often geometrically close together, but not always. Purity metrics are listed.

Question 8.

The true classes, using both NMF and LSI embeddings to $\mathbb{R}^{20}$ (before the PCA embedding in $\mathbb{R}^2$), do not lie exactly in a well-defined region with "hard" boundaries. That is, in the right-hand column of Figure 6, the class labels are not concentrated in a convex region. This may make the data (or its TF-IDF representation) somewhat ill-suited for K-Means clustering, though the purity metrics show that the clusters do capture the class patterns to an extent.

The week 2 lecture notes note several difficulties for clustering: "anisotropicly distributed blobs", "unequal variance", and "unevenly sized blobs" are exhibited in the data, whereas "incorrect number of blobs", by the design of this task (2 clusters for 2 classes), is not exhibited.

On the other hand, K-Means partitioning as a rule creates well-defined regions with hard boundaries, and the K-Means region boundaries seem to fall close to the true class's "soft" boundaries to the extent that they exist. NMF seems to perform worse than LSI, and the performance metrics in Figure 6 mostly confirm this analysis.

Question 9.

```
11314 Samples, 20 Classes, 20 Clusters, 20 Components

Clustering Performance After LSI Dimensionality Reduction:
{'adjusted_rand_score': 0.12658561875449764,
 'completeness_score': 0.3735937287180772,
 'homogeneity_score': 0.3337776185521292,
 'mutual_info_score': 0.998079219619663,
 'v_measure_score': 0.35256510051967754}

Clustering Performance After NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.08630783201129401,
 'completeness_score': 0.37269740057340445,
 'homogeneity_score': 0.30470709659852174,
 'mutual_info_score': 0.9111510307517168,
 'v_measure_score': 0.3352901945961416}
```

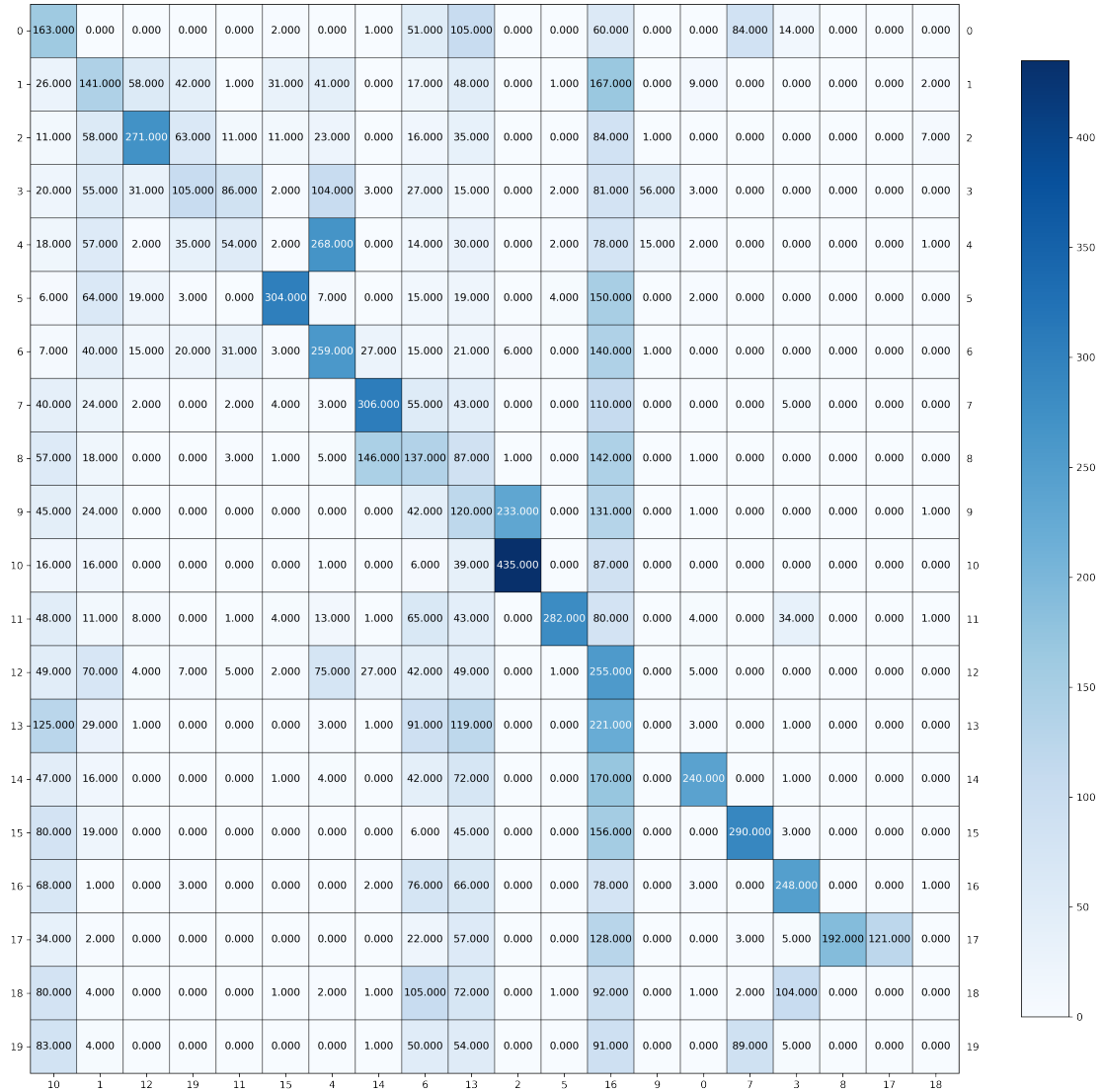


Figure 7: Permuted contingency matrix of SVD case performed K-mean clustering with 20 components ("Rows" of the matrix are the actual classes and "Columns" are the clusters).

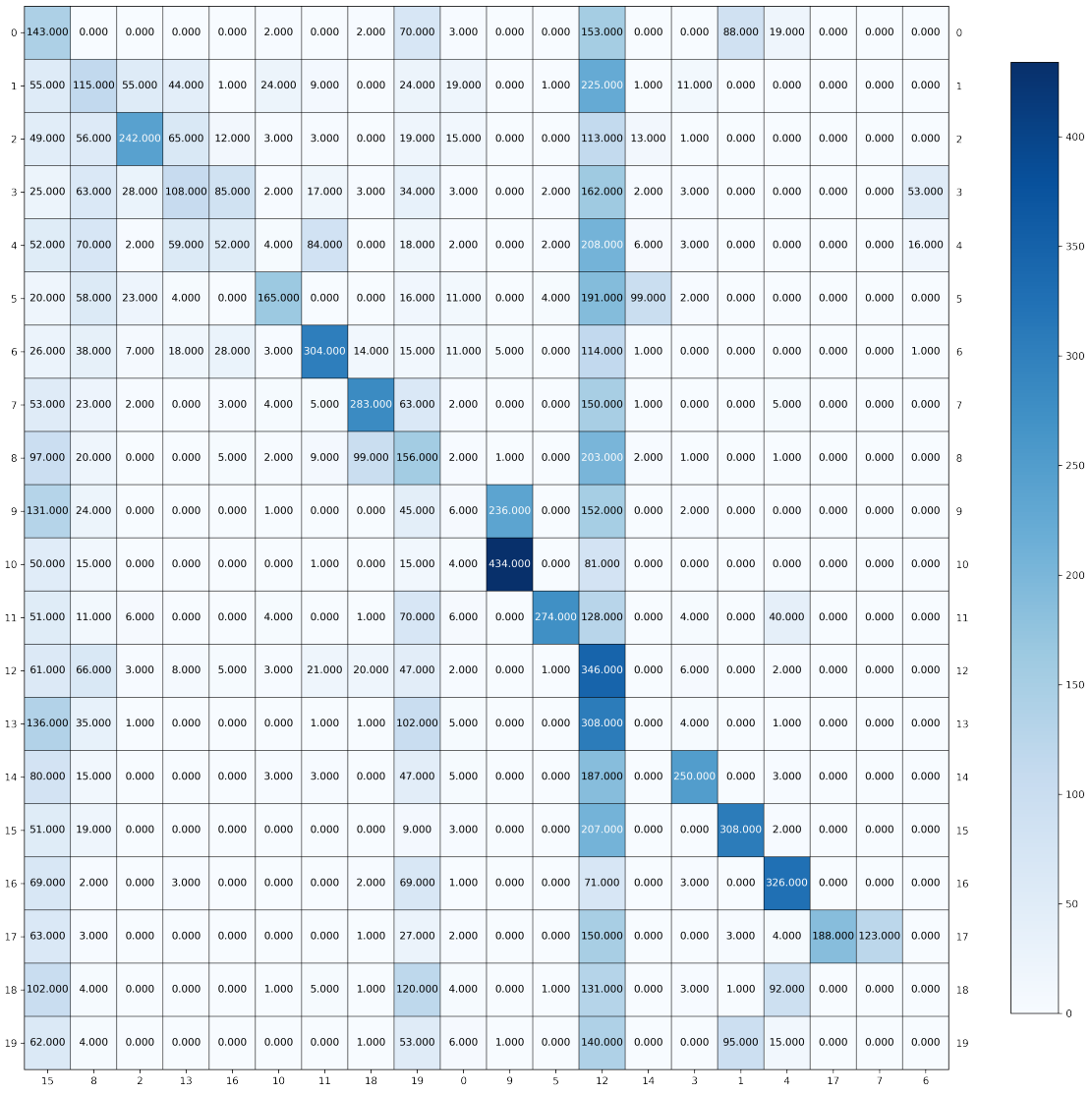


Figure 8: Permuted contingency matrix of NMF case performed K-mean clustering with 20 components ("Rows" of the matrix are the actual classes and "Columns" are the clusters).

Both NMF and SVD embeddings have quite similar scores in the five metrics, but SVD is slightly better in the adjusted rand score. This result is logically consistent with the answer to question 7. In the contingency matrix of the two cases lie with each other. Some classes like 11th(rec.sport.hockey) and 6th(comp.windows.x) are clustered well, but 19th(talk.politics.misc) and 20th(talk.religion.misc) classes are matched with 0 percent accuracy. Graphically, this figure may show how K-mean falls into local optimal points with reduction and how the matching precision depends on classes. Intuitively, the overall precision of these figures tells us we need to extend both SVD and NMF embeddings.

Question 10.

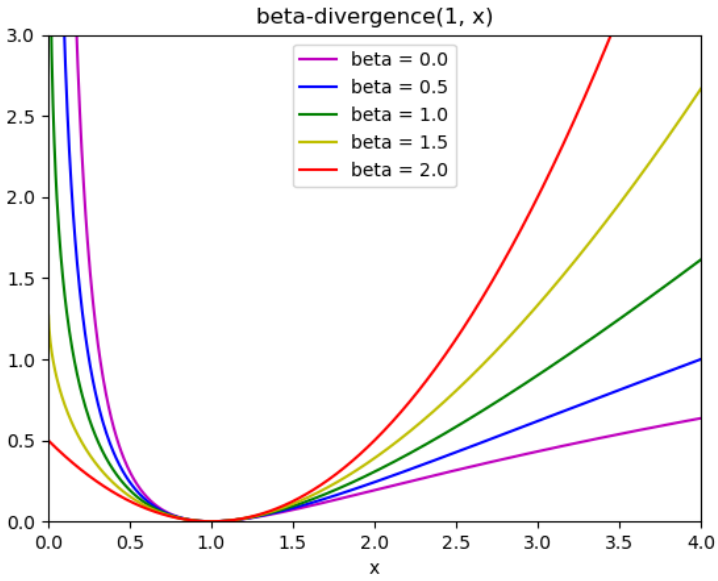


Figure 9: Beta-divergence loss functions with $beta = 1$ is "Kullback-Leibler" and $beta = 2$ is "Frobenius norm"

```
11314 Samples, 20 Classes, 20 Clusters

Clustering Performance After LSI Dimensionality Reduction:
{'adjusted_rand_score': 0.12658561875449764,
 'completeness_score': 0.3735937287180772,
 'homogeneity_score': 0.3337776185521292,
 'mutual_info_score': 0.998079219619663,
 'v_measure_score': 0.35256510051967754}

Clustering Performance After KL_NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.2676535269815391,
 'completeness_score': 0.45653052363109853,
 'homogeneity_score': 0.44447811412275146,
 'mutual_info_score': 1.3291016072498292,
 'v_measure_score': 0.4504237088978026}

Clustering Performance After FB_NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.08650669788541554,
 'completeness_score': 0.37522850442236716,
 'homogeneity_score': 0.3054007263279406,
 'mutual_info_score': 0.9132251584959508,
 'v_measure_score': 0.33673269560642066}
```

Kullback-Leibler Divergence of our NMF model noticeably enhanced in all five scores by approximately 30% to 300%. Unlike the default Frobenius norm, Kullback-Leibler Divergence as known as relative entropy gives less penalty to the gap between X and WH distribution. This lenient in the penalty function may help our same model avoid local optimal positions.

- β-divergence loss function is calculated as

$$d_\beta(X, Y) = \sum_{i,j} \frac{1}{\beta(\beta-1)}(X_{ij}^\beta + (\beta-1)Y_{ij}^\beta - \beta X_{ij} Y_{ij}^{\beta-1})$$

Question 11.

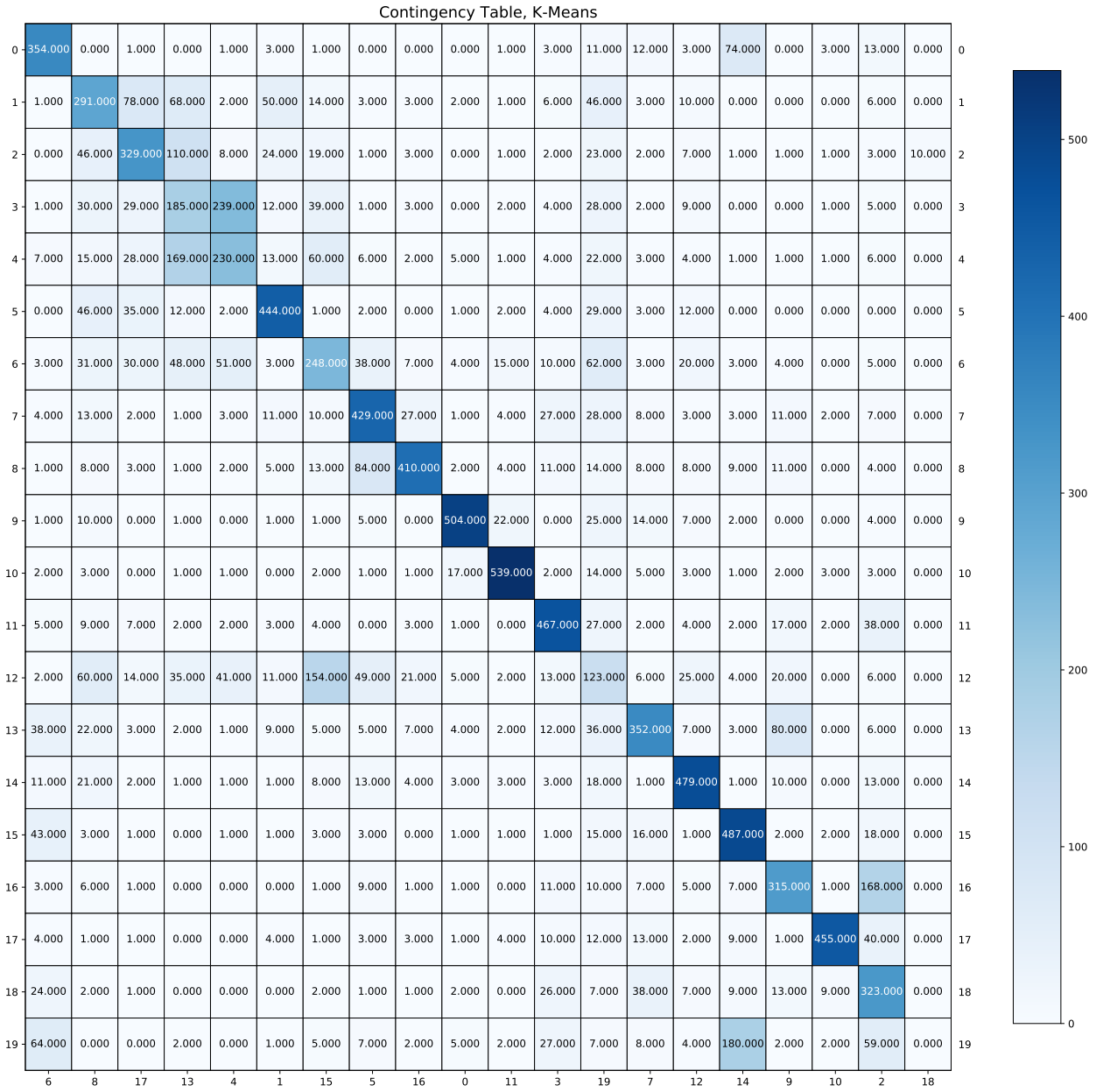


Figure 10: Permuted contingency matrix of UMAP reduction with "cosine" metric applied K-mean clustering with 120 components.

```
11314 Samples, 20 Classes, 20 Clusters, 120 Components

Clustering Performance After UMAP_cosine Dimensionality Reduction:
{'adjusted_rand_score': 0.43223618599817903,
 'completeness_score': 0.5873927721983061,
 'homogeneity_score': 0.5559774529637388,
 'mutual_info_score': 1.6625127376343567,
 'v_measure_score': 0.5712535278410621}
```

- n components for the grid search is

$$[2, 5, 10, 20, 30, 40, 60, 80, 120, 160, 320]$$

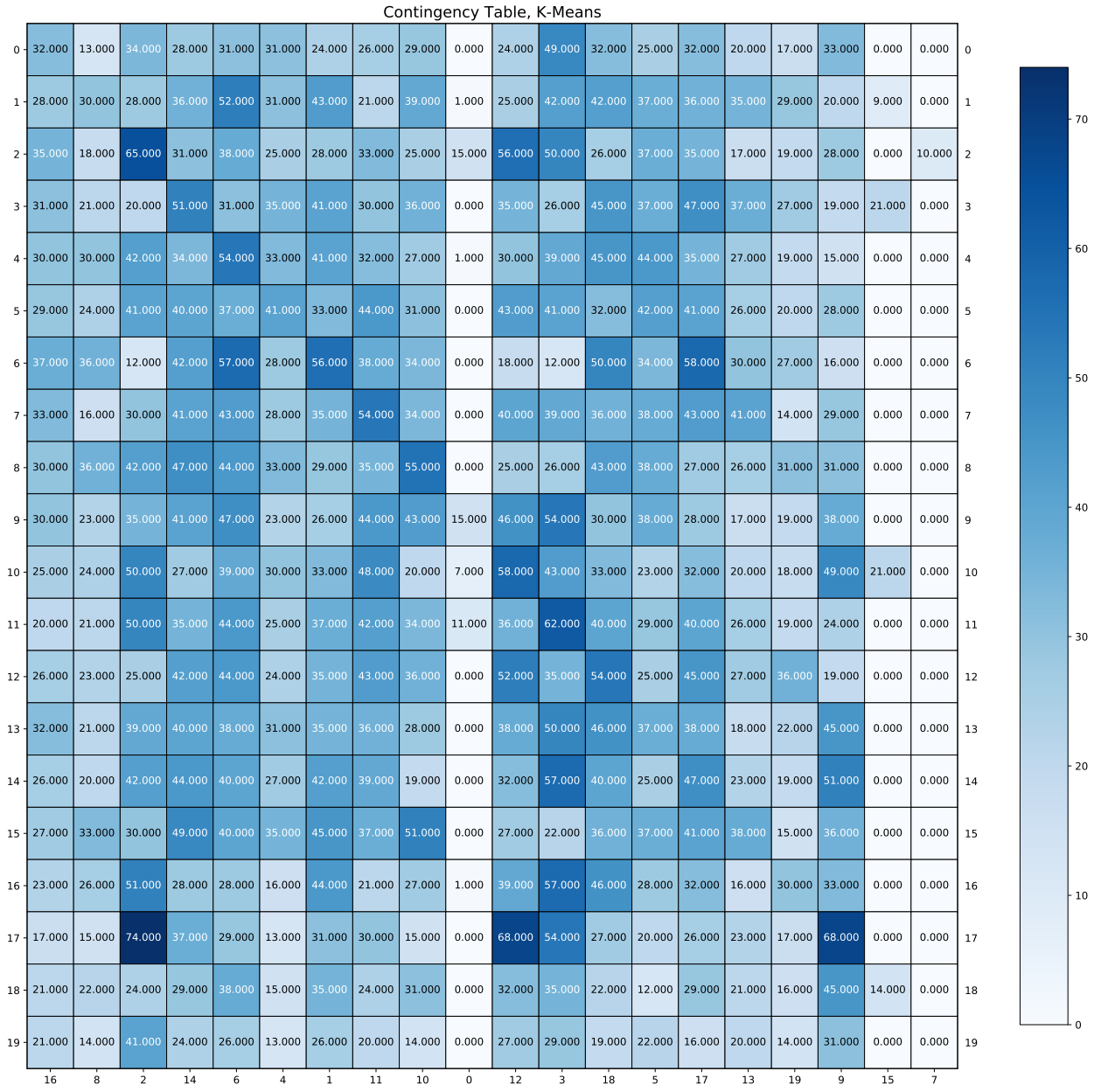


Figure 11: Permuted contingency matrix of UMAP reduction with "euclidean" metric applied K-mean clustering with 120 components.

```
11314 Samples, 20 Classes, 20 Clusters, 120 Components

Clustering Performance After UMAP_euclidean Dimensionality Reduction:
{'adjusted_rand_score': 0.0028777633923741408,
 'completeness_score': 0.017848956758723857,
 'homogeneity_score': 0.01693530512066055,
 'mutual_info_score': 0.050640831437923084,
 'v_measure_score': 0.017380131844833866}
```

We've conducted a grid search to find the best components for UMAP. For both Euclidean and Cosine, the five scores maximized at 20 and 120 components. Although both are slightly different, the highest scores appeared at 120 components. Then, we used 120 components to conduct comparing Euclidean and Cosine metric. We have analyzed the differences between both metrics in question 12.

Question 12.

UMAP builds mathematical theory to justify the graph-based approach. Before performing manifold, our UMAP reduction takes a distance that is defined with Cosine or Euclidean. Over the contingency matrices and scores, their effects are quite the opposite. It seems that only cosine similarity distance works besides Euclidean does not.

In a text distancing sense, the length of text data affects the scores of the result. For example, we assume the word 'sports' occurs more in long science documents, Euclidean metric could rate these documents more of the sport than shorter sports documents. It could be a reason for the low numbers in five scores. Unlike Euclidean, the Cosine metric is independent of the length of documents. Thus the cosine could have shown the best scores in all five scores and the contingency matrix.

Let's take a look inside the Cosine contingency matrix. The first thing the Cosine UMAP hardly classifying is the 4th and 5th classes, which are "comp.sys.ibm.pc.hardware" and "comp.sys.mac.hardware". Obviously, the results make sense. The interesting thing is that our classifier thought the half of the "sci.electronics" to "misc.forsale". It is also reasonable that we, occasionally confusing to discriminating electronics advertisements from just electronics documents. The last thing we have seen on the matrix is that the classifier totally failed to find "talk.religion.misc". Mostly, it misunderstood "talk.religion.misc" to "soc.religion.christian" and the result fairly make sense.

Question 13.

We compare the evaluation metrics for Agglomerative clustering after Ward Dimensionality Reduction and Single Dimensionality Reduction as shown in figure 12.

```
11314 Samples, 20 Classes, 20 Clusters

Clustering Performance After Ward Dimensionality Reduction:
{'adjusted_rand_score': 0.44757449615298284,
 'completeness_score': 0.5771678350630784,
 'homogeneity_score': 0.5636140135267851,
 'mutual_info_score': 1.6853479787760666,
 'v_measure_score': 0.5703104066751674}

Clustering Performance After Single Dimensionality Reduction:
{'adjusted_rand_score': 0.42310117529009406,
 'completeness_score': 0.578004548500406,
 'homogeneity_score': 0.5583776887216022,
 'mutual_info_score': 1.6696900476124892,
 'v_measure_score': 0.5680216273904627}
```

Agglomerative clustering is a bottom-up hierarchical clustering algorithm. The result forms a tree grown from the bottom is called a dendrogram which is a goal of clustering. We used two linkage methods to measure the similarity between clusters. Single linkage computes all pairwise between elements in the clusters and chooses the smallest of these values. On the other hand, Ward's minimum variance linkage minimizes the total variance of the inside of each cluster. Ward criteria conduct more compact clustering than Single linkage. In our reduction with Ward and Single, both five scores are quite high but almost similar. This shows in any of the two linkage criteria, Agglomerative clustering is as effective as K-mean clustering with Cosine UMAP.

Question 14.

We performed grid search on three parameters for HDBSCAN. Those parameters are minimum samples ranging from 0 to 60, alpha ranging from 0.2 to 2 and metric ('euclidean', 'l1', 'l2'). We also performed grid search on 2 parameters of DBSCAN, those parameters are eps ranging from 0.1 to 1.5 and metric ('euclidean', 'l1', 'l2'). We found that DBSCAN is the best performing model by comparing the evaluation metrics as shown in figure 13.

```
11314 Samples, 20 Classes, 20 Clusters

Clustering Performance After UMAP Dimensionality Reduction with Parameters:

eps: 0.6; metric: euclidean
{'adjusted_rand_score': 0.2790436475095547,
 'balanced_accuracy_score': 0.048522872254845556,
 'completeness_score': 0.5908797644854894,
 'homogeneity_score': 0.46221142510341195,
 'mutual_info_score': 1.3821286773740238,
 'v_measure_score': 0.5186851446629649}
```

Figure 12: Performance metrics for DBSCAN clustering with $k = 20$ clusters and 20 true classes.

Question 15.

The DBSCAN is the best clustering model, its contingency matrix is shown in figure 14. There are 14 clusters given by the model. The DBSCAN works by finding core samples of high density and expands clusters from them. The number of clusters it will find depend on the number of samples in a neighborhood for a point to be considered as a core point. The minimum cluster size is 100 for the current model, which is why the number of clusters is smaller than the number of classes.The model tends to assign class 3,4 6 and 12 as cluster 1. and class 1,2,5 as cluster 3. The clustering label "-1" corresponds to the last column in the contingency table. Those points are noise in the data.

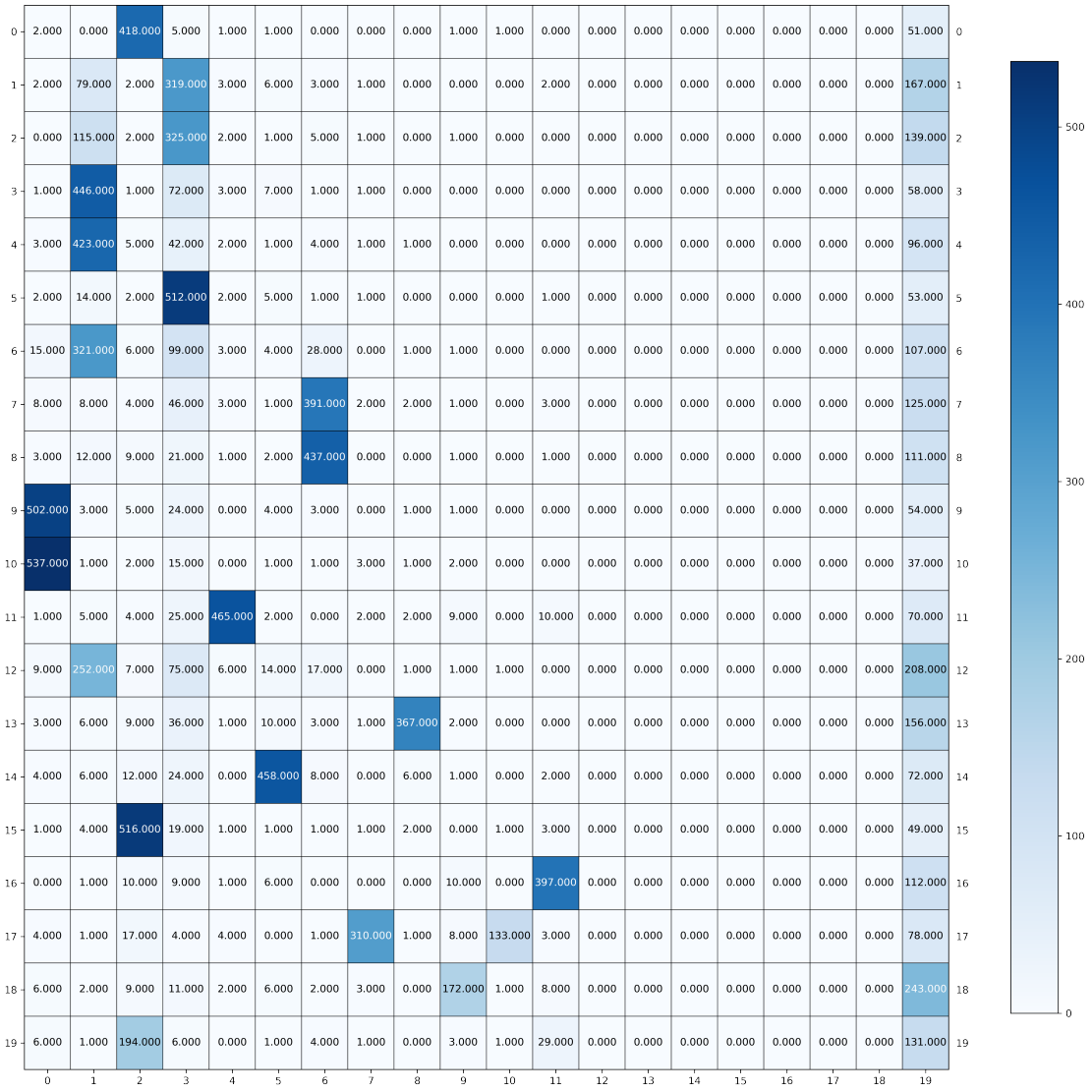


Figure 13: Contingency table for DBSCAN model.

Question 16.

We first acquired the data from the Kaggle website and downloaded it into our directory.

```
train = pd.read_csv("input/BBC News Train.csv",header=0)
test = pd.read_csv("input/BBC News Test.csv")
```

We then analyzed the beginning of the data to see how we would need to shape it. By running the command,

```
train.head()
```

we got the following result.

```
   ArticleId                                               Text  Category
0       1833  worldcom ex-boss launches defence lawyers defe...  business
1        154  german business confidence slides german busin...  business
2       1101  bbc poll indicates economic gloom citizens in ...  business
3       1976  lifestyle  governs mobile choice  faster  bett...      tech
4        917  enron bosses in 168m payout eighteen former e...  business
```

Based off of this output, we realized that the category is in text so to properly compare it to the result from our clustering, we need to set up a numerical labeling system. We did this by running the following code

```
train['category_id'] = train['Category'].factorize()[0]
```

This makes another category in the data frame that will give data from the same category the same number. To check if this is correct, we ran the following command:

```
train.head(20)
```

and found the following result.

```
    ArticleId                                               Text       Category  category_id
0        1833  worldcom ex-boss launches defence lawyers defe...       business            0
1         154  german business confidence slides german busin...       business            0
2        1101  bbc poll indicates economic gloom citizens in ...       business            0
3        1976  lifestyle  governs mobile choice  faster  bett...           tech            1
4         917  enron bosses in 168m payout eighteen former e...       business            0
5        1582  howard  truanted to play snooker  conservative...       politics            2
6         651  wales silent on grand slam talk rhys williams ...          sport            3
7        1797  french honour for director parker british film...  entertainment            4
8        2034  car giant hit by mercedes slump a slump in pro...       business            0
9        1866  fockers fuel festive film chart comedy meet th...  entertainment            4
10       1683  blair rejects iraq advice calls tony blair has...       politics            2
11       1153  housewives lift channel 4 ratings the debut of...  entertainment            4
12       1028  uk coal plunges into deeper loss shares in uk ...       business            0
13        812  bp surges ahead on high oil price oil giant bp...       business            0
14        707  ireland 21-19 argentina an injury-time dropped...          sport            3
15       1588  wenger signs new deal arsenal manager arsene w...          sport            3
16        342  u2 s desire to be number one u2  who have won ...  entertainment            4
17        486  hantuchova in dubai last eight daniela hantuch...          sport            3
18       1344  melzer shocks agassi in san jose second seed a...          sport            3
19       1552  moving mobile improves golf swing a mobile pho...           tech            1
```

From this table, we can see our code successfully worked. We also note that there are 5 difference categories that the data is separated into. Thus, for the K-Means algorithm later we will use 5 categories. Before working on the data further, we plotted a bar graph to get a sense of the distribution of the data to make sure it is even.

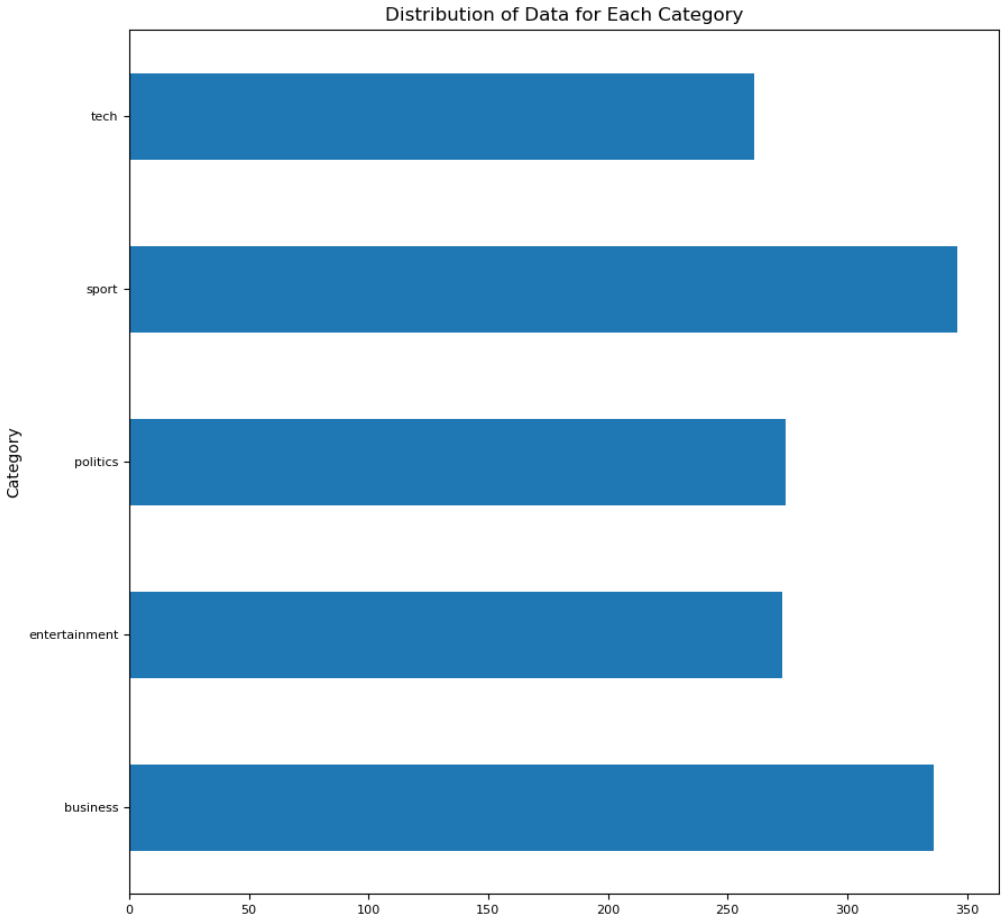


Figure 14: Bar graph showing the distribution of the 5 different categories the data is separated into.

From the bar graph, we can see that the data is well balanced. We then stored the data and our newly created labels and then transformed the documents into TF-IDF vectors using min_df=3. We also did not perform any stemming or lemmatization. The dimensions of the TF-IDF matrix that we created were found to be (1490, 10079) as shown in the console

```
(1490, 10079)
```

Now we followed the steps detailed in Part (1) to find a good clustering of the data. First, we tried to find the effective dimension of the data through inspection of the top singular values of the TF-IDF

matrix. We did that by following Question 4 and creating a plot of the percent of variance the top r principle components can retain v.s. r. We generated the following plot.

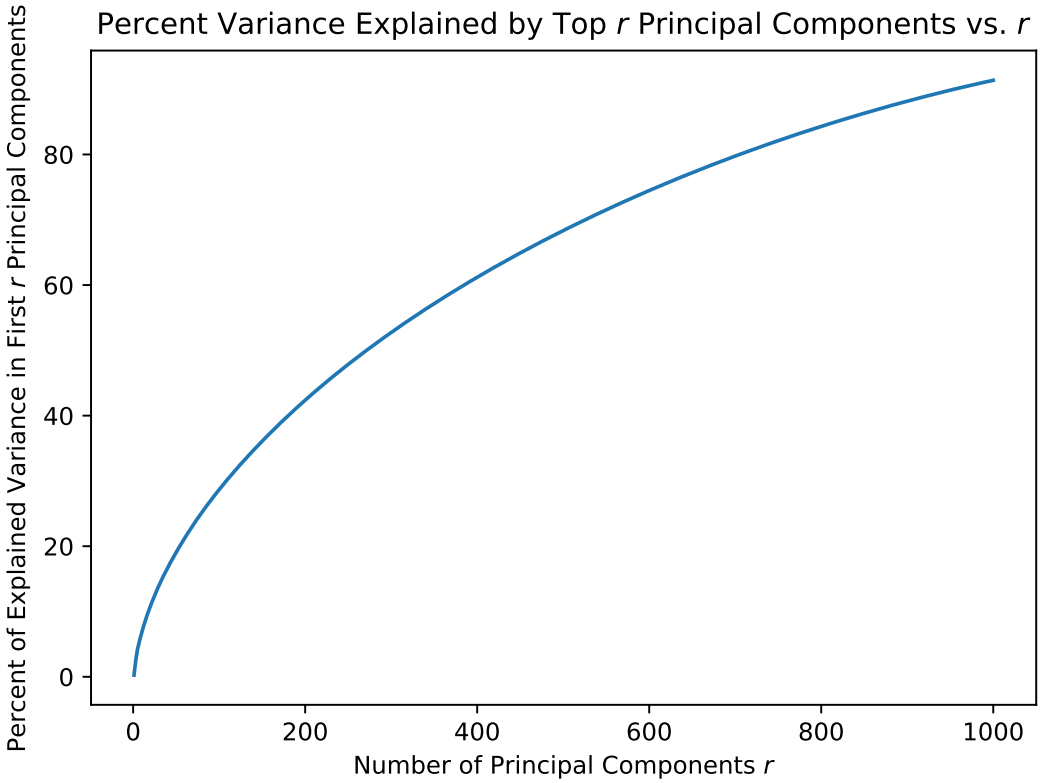


Figure 15: Plot of the variance of the top r principle components can retain v.s. r for r=1 to 1000.

From the figure above, we can see that up to 80 percent of the variance of the original data is retained in the first 1000 principle components meaning that an "implicit vocabulary" of around 1000 elements is quite effective at describing the articles well. Following the steps in Question 5, we then do a grid search to choose between Truncated SVD / PCA and NMF and r, the dimension that we want to reduce the data to. To better visualize this, we generated a plot of the 5 measure scores v.s. r for both SVD and NMF.

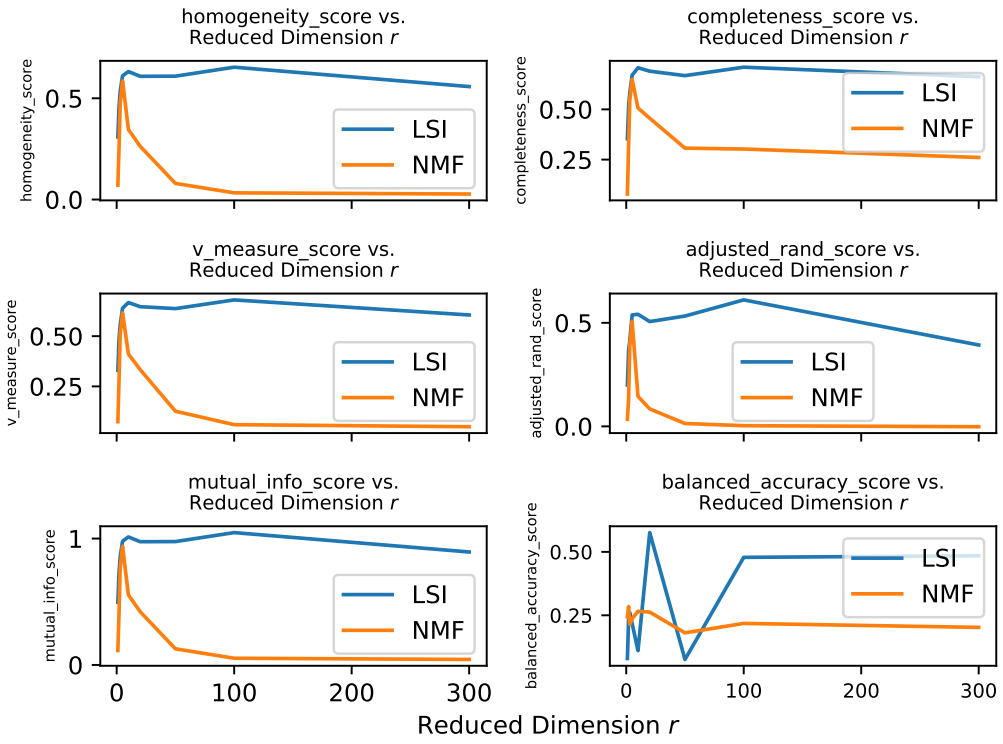


Figure 16: Plot of the 5 measure scores v.s. r for both SVD and NMF.

From this figure, it seems the best possible performance occurs at lower values of r for both LSI and NMF but mainly NMF. We generated a figure with a closer look at those r values as shown below.

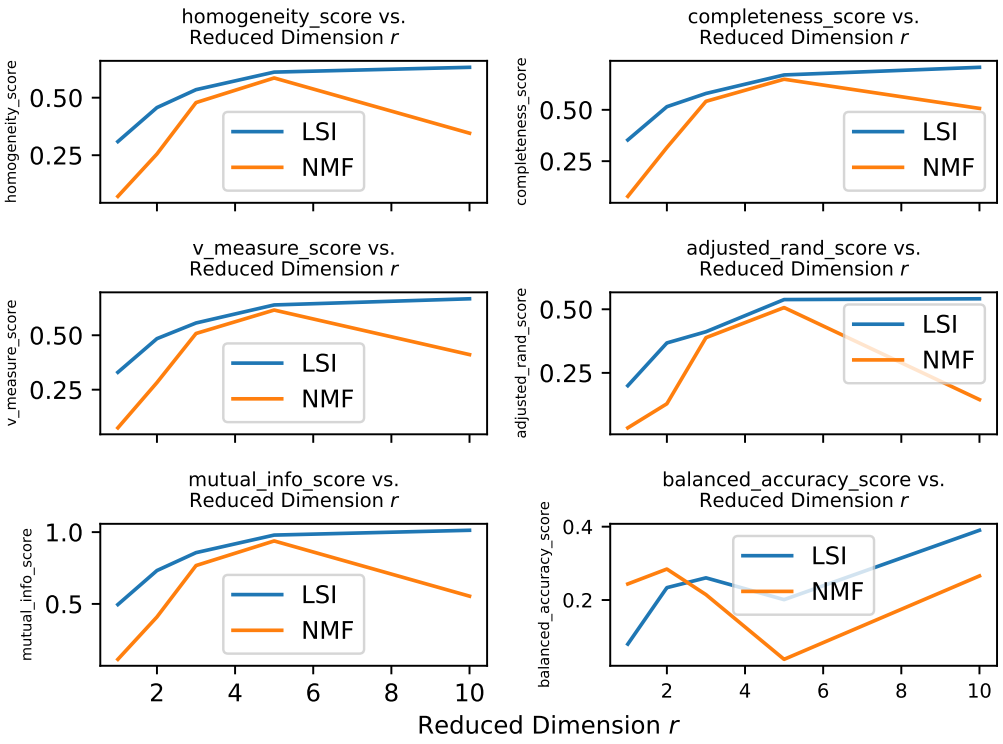


Figure 17: A more focused plot of the 5 measure scores v.s. r for both SVD and NMF.

From this plot, it appears that use LSI with reduced dimension of r = 5 is the best choice. We then followed the by visualizing the clustering results for SVD with our choice of r = 5 and NMF with our choice of r = 5. We found the following result.

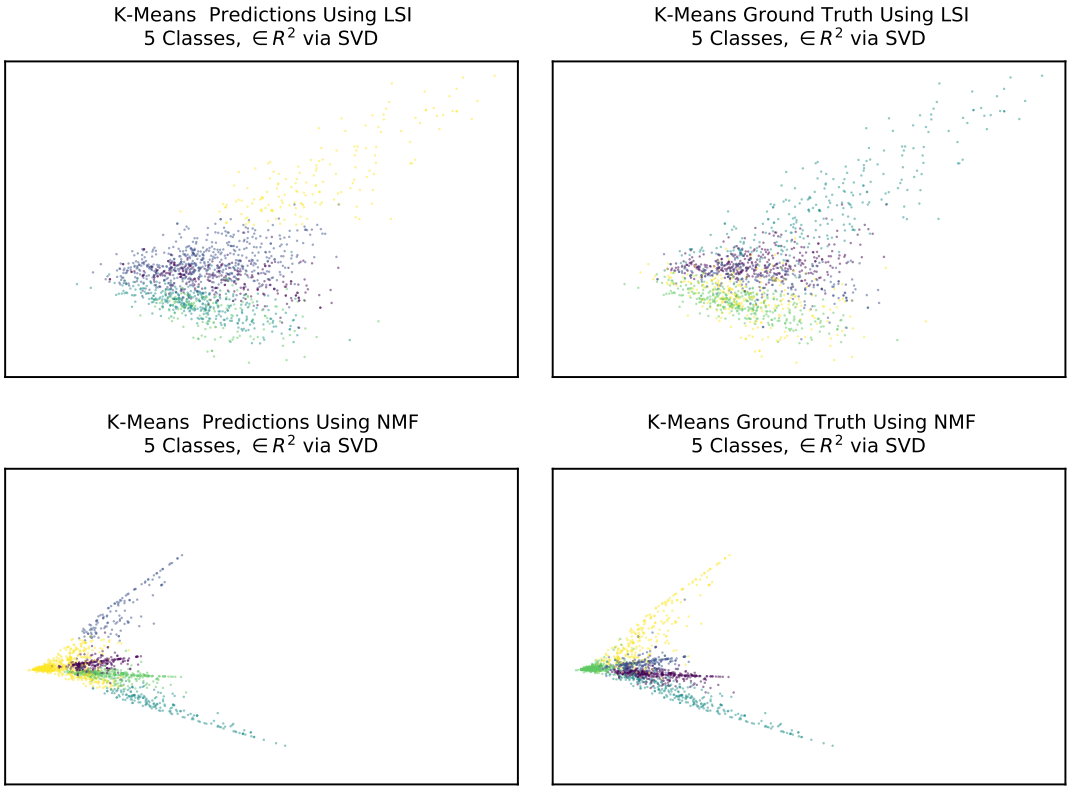


Figure 18: Visualization of our clustering results for SVD and NMF with our choice of r=5.

From this depiction, we find that in both cases clustering might not work extremely well because of the overlap between data points suggesting the problem of "incorrect number of blobs". Visually, there does not seem a good way for them to be distinctly separated as points from each category overlap. In addition, in the NMF plots, we see the problem of anisotropicaly distributed blobs. Another problem that we see is the problem of unequal variance for both LSI and NMF. For LSI, we see that in the cluster in the upper-right hand corner. For NMF, we see that in the top and bottom clusters that stretch outwards. We then ran K-Means on the LSI and NMF dimensionally reduced data with our choice of r=5. Our results are as follows:

```
Clustering Performance After LSI Dimensionality Reduction:
{'adjusted_rand_score': 0.563074920791721,
 'balanced_accuracy_score': 0.026245412509759713,
 'completeness_score': 0.6863908542475007,
 'homogeneity_score': 0.6293445308581864,
 'mutual_info_score': 1.008468388251016,
 'v_measure_score': 0.6566310141716596}

Clustering Performance After NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.5066953868613383,
 'balanced_accuracy_score': 0.2364247552235219,
 'completeness_score': 0.6489833136270141,
 'homogeneity_score': 0.5856959160182447,
 'mutual_info_score': 0.938525382315934,
 'v_measure_score': 0.6157176167360328}
```

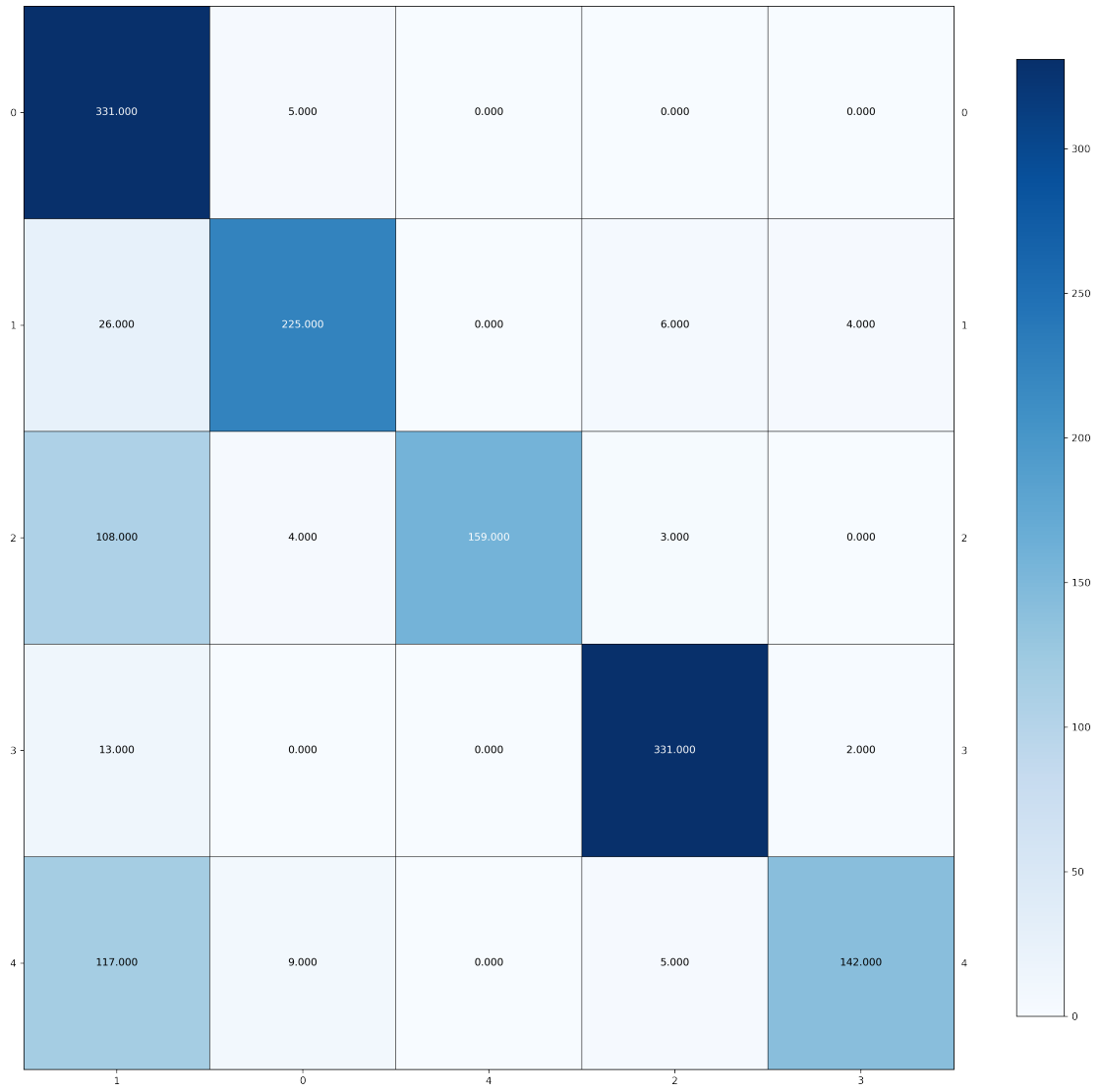


Figure 19: Contingency Matrix from K-Means using LSI with r = 5 for dimensionality reduction.

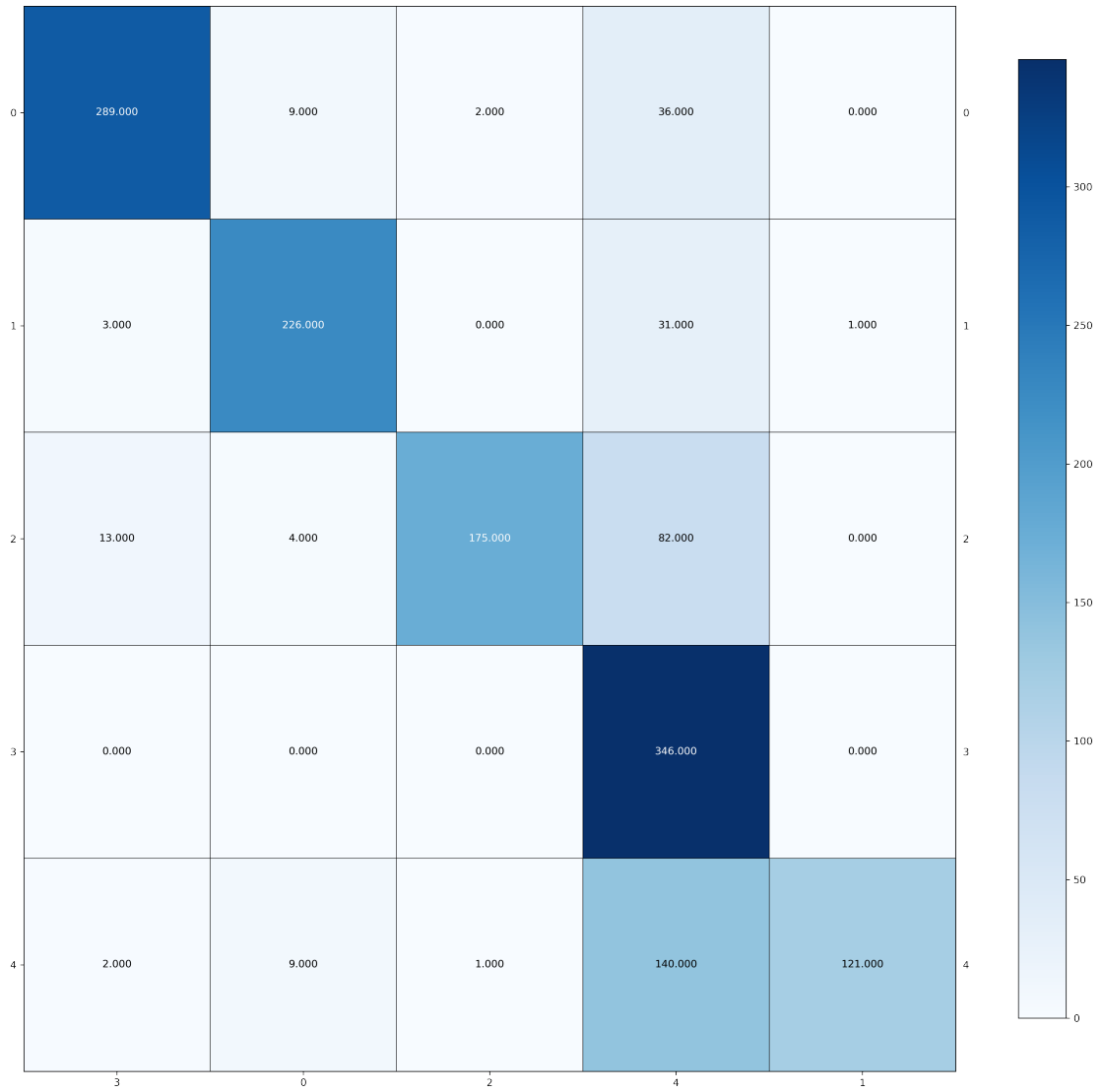


Figure 20: Contingency Matrix from K-Means using NMF with r = 5 for dimensionality reduction.

While these results are not the best they are acceptable. The homogeneity score, completeness score, and v measure score for NMF Dimensionality reduction are all in the high 50 and low 60 percents while for LSI they are in between 60 and 70 percent. Thus, the LSI Dimensionality reduction seemed to have perform better when applying it onto data for the K-Means algorithm. In addition, based off of the contingency matrix, we can still clearly see that there are many mislabelings with LSI struggling with category 1 and NMF struggling with category 4. We continue to follow the procedure from Part 1 by using Kullback-Leibler Divergence for NMF with our choice of r = 5 and then performing K-Means clustering. We got the following results

```
Clustering Performance After KL_NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.860653095682575,
 'balanced_accuracy_score': 0.19504165619075656,
 'completeness_score': 0.8328000978296903,
 'homogeneity_score': 0.829462977945581,
 'mutual_info_score': 1.329140322141225,
 'v_measure_score': 0.8311281881263124}
```

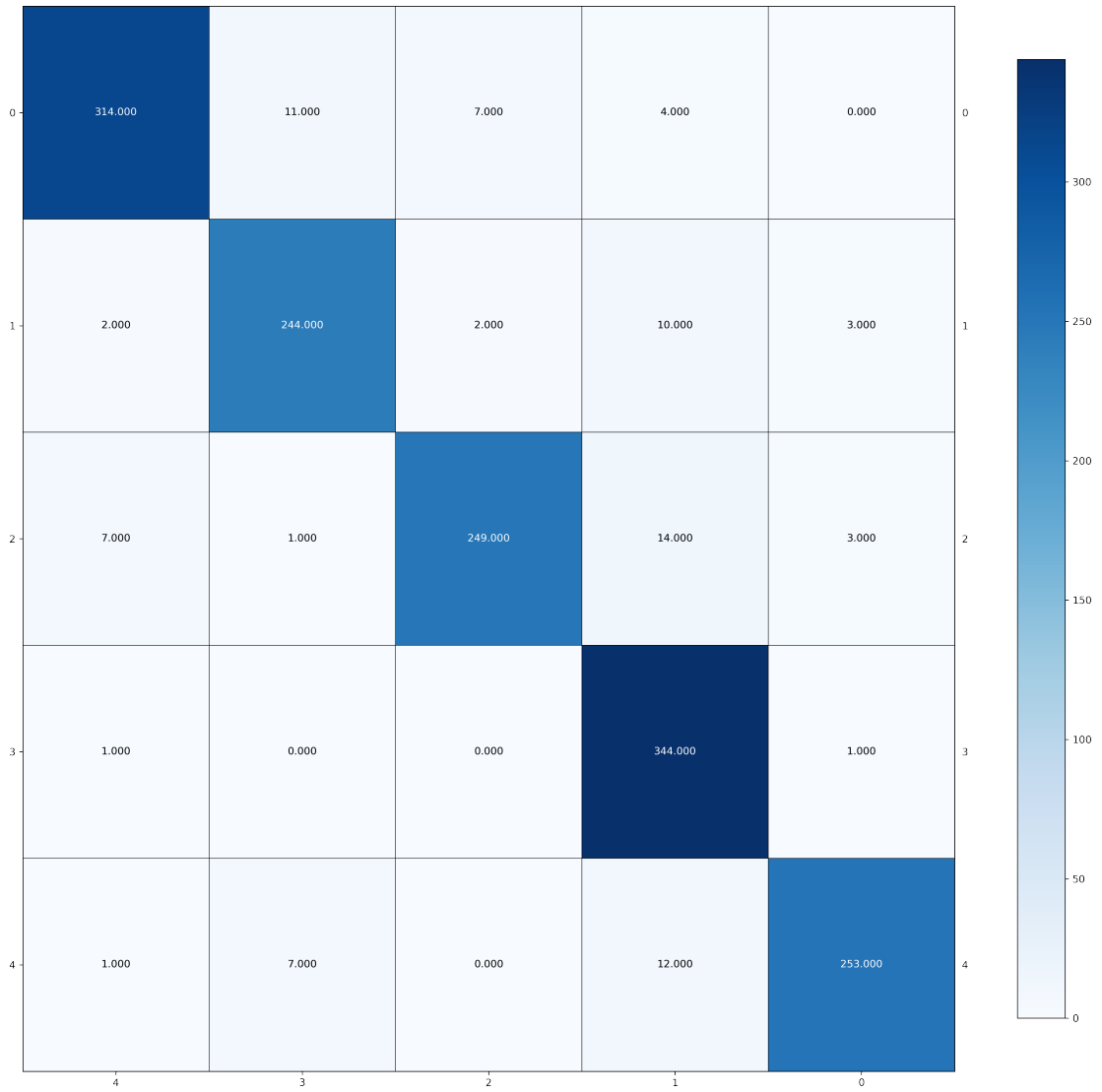


Figure 21: Contingency matrix of K-Means Algorithm after using KL-NMF with n_components=5.

This has given us a remarkable improvement over using using LSI or NMF as seen in all metrics and in the contingency matrix where there are significantly fewer incorrect categorizations. Continuing further on the procedure from Part 1, we know try a different dimensionality reduction tool called UMAP on the data followed by using the K-means algorithm. Before we can use this tool, we first need to find a good n_component and determine whether the Euclidean or Cosine metric has a better performance. We did this through a grid search and found that using n_component=40 and a metric of Cosine gave us the best performance. The results for this can be seen below

```
Clustering Performance After UMAP cosine Dimensionality Reduction:
{'adjusted_rand_score': 0.8127033624389322,
 'balanced_accuracy_score': 0.18480699300177675,
 'completeness_score': 0.7760084068416585,
 'homogeneity_score': 0.7758650981747077,
 'mutual_info_score': 1.2432545079712054,
 'v_measure_score': 0.7759367458912217}
```

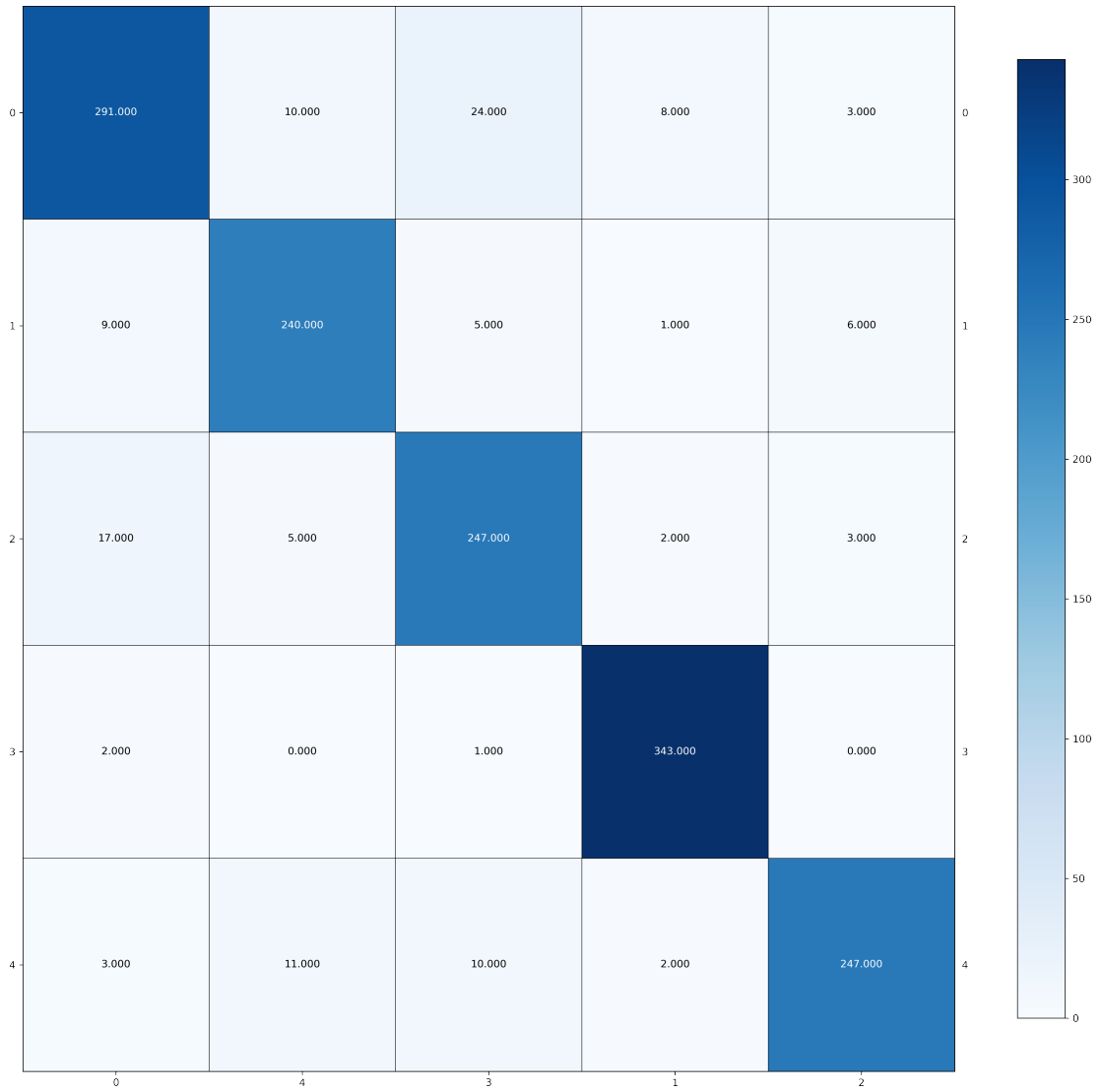


Figure 22: Contingency matrix of K-Means Algorithm after using UMAP with n_components=40 and "metric"=cosine.

This result was not as satisfactory as before as all metrics are lower than using KL-NMF Dimensionality Reduction and there are more mismatches in the contingency matrix. However, the results are still very good. Following Part 1, we then tried Agglomerative Clustering and compared the performance of "ward" and "single". The results are as follows.

```
Clustering Performance After Ward Dimensionality Reduction:
{'adjusted_rand_score': 0.3545803605717456,
 'balanced_accuracy_score': 0.060449252693778245,
 'completeness_score': 0.4573187317551887,
 'homogeneity_score': 0.8159235092099191,
 'mutual_info_score': 1.3074445330398103,
 'v_measure_score': 0.5861211518686157}

Clustering Performance After Single Dimensionality Reduction:
{'adjusted_rand_score': 0.47781790890802855,
 'balanced_accuracy_score': 0.3081801350555834,
 'completeness_score': 0.6074274369276712,
 'homogeneity_score': 0.6065048666353688,
```

```
 'mutual_info_score': 0.9718698667137414,
 'v_measure_score': 0.606965801211747}
```

The metrics (especially the completeness core, homogeneity score, and v measure score) for both "ward" and "single" using Agglomerative clustering are outright worse than K-Means Algorithm after using KL-NMF and K-Means Algorithm after using UMAP. Finally, we test DBSCAN and HDBSCAN. As we did before, we perform a grid search. For DBSCAN, we iterate over the parameter eps by testing the values $0.1, 0.2, ..., 1.5$ and also iterate over the parameter metric by trying cosine, euclidean, l1, and l2. For HDBSCAN, we iterate over the parameters min_samples for None, 1, 2, 3, 5, 10, 20, 40, 60, alpha for .2,.4,.6,...,2.0, and metric for euclidean, l1, and l2. We keep the min_cluster_size=100 for HDBSCAN and min_samples=100 for DBSCAN. Our reasoning is that there are enough samples in each category and we would be tuning the min_samples in HDBSCAN and eps parameter in DBSCAN anyways.

For DBSCAN, we found the following parameters and metrics gave the best result

```
Clustering Performance After UMAP Dimensionality Reduction with Parameters:

eps: 1.0; metric: l2
{'adjusted_rand_score': 0.6286230891952207,
 'balanced_accuracy_score': 0.30373796347749504,
 'completeness_score': 0.6293832203554605,
 'homogeneity_score': 0.6977533387397528,
 'mutual_info_score': 1.118087391585204,
 'v_measure_score': 0.6618071672280618}
```

For HDBSCAN, we found the following parameters and metrics gave the best result

```
Clustering Performance After UMAP Dimensionality Reduction with Parameters:

min_samples: 40; alpha: 2.0; metric: euclidean
{'adjusted_rand_score': 0.5660542468970623,
 'balanced_accuracy_score': 0.3779391847350569,
 'completeness_score': 0.8046207317655343,
 'homogeneity_score': 0.5607267776419658,
 'mutual_info_score': 0.8985145686841609,
 'v_measure_score': 0.6608901939442469}
```

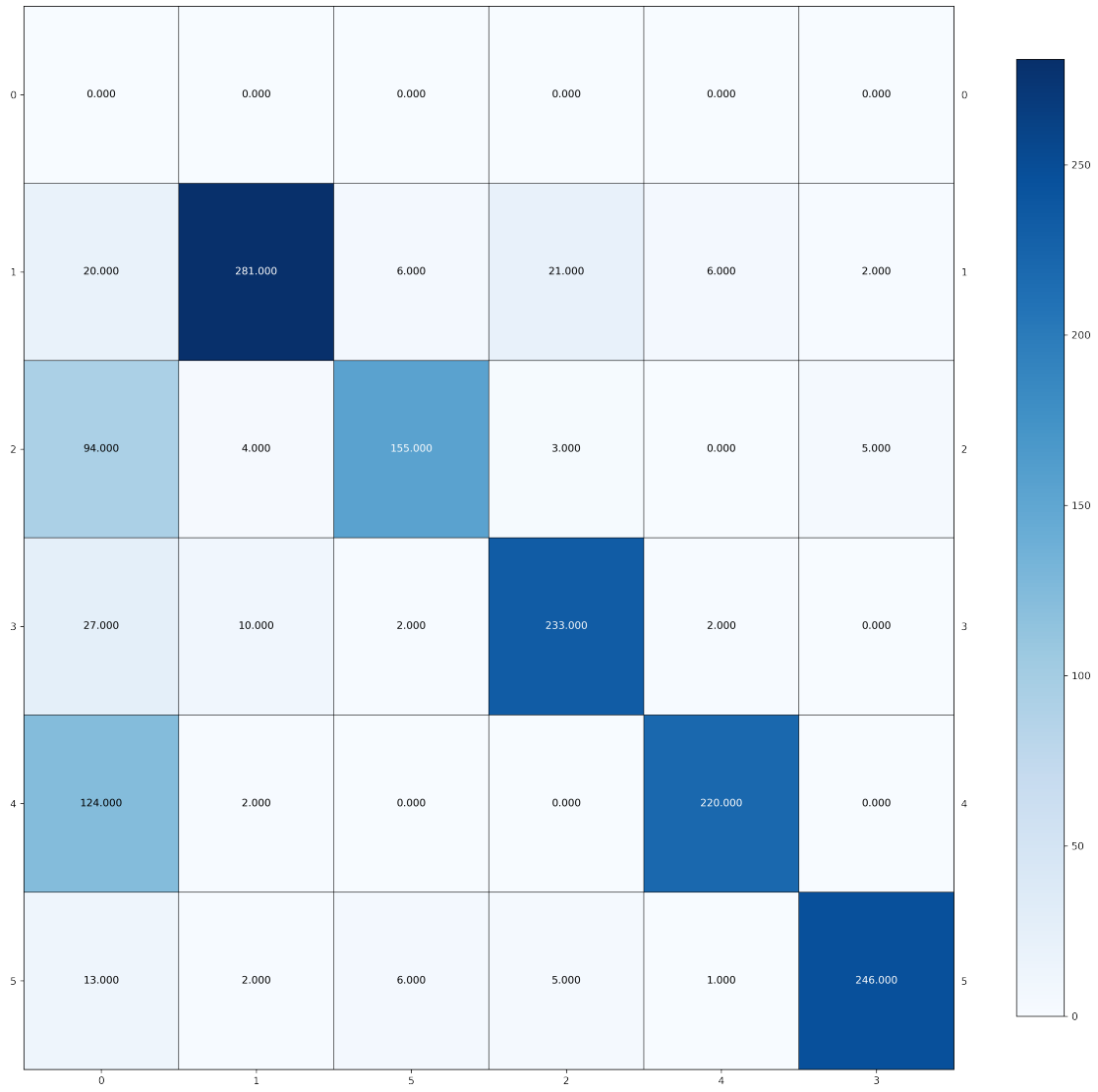


Figure 23: Contingency matrix of DBSCAN with eps: 1.0, metric:l2.

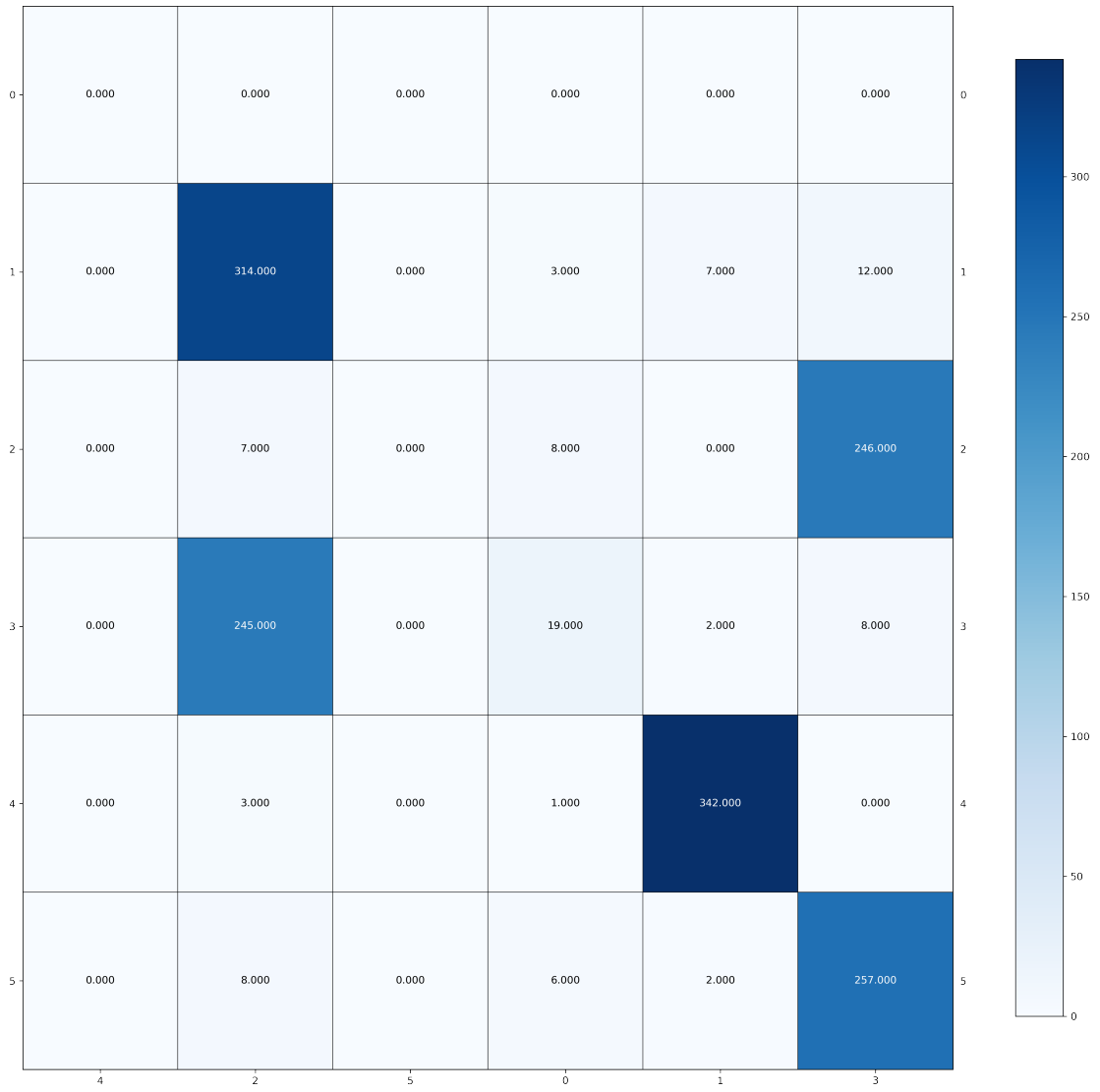


Figure 24: Contingency matrix of HDBSCAN with min_samples: 40; alpha: 2.0, metric: euclidean.

The performance of both are a bit difficult to compare in terms of metrics because each performs better on different metrics. DBSCAN performs better on homogeneity, slightly better on v measure, better on adjusted Rand Index, and the mutual info score. HDBSCAN performs much better on the completeness score. However, both cases still fail to perform better than K-Means after using KL-NMF with n_components=5. However, when looking at the contingency matrices, the contingency matrix with DBSCAN looks better with more higher values along the diagonal. Meanwhile the contingency matrix of HDBSCAN has a severe issue in classifying two of the categories seeing as how two columns have almost no values. This seems to suggest that HDBSCAN formed three clusters rather than desired five. However, we reiterate that both are much worse than the K-Means Algorithm after using KL-NMF with n_components=5 on the data.

In conclusion, using the K-Means Algorithm after using KL-NMF with n_components=5 on the data gave us the best result on the BBC data set. This was determined by the clearly superior 5 metrics of homogeneity, completeness, v-measure, adjusted Rand Index, and adjusted mutual information score as

well as a confusion matrix that showed very few mismatches and errors. We repeat those metrics below

```
Clustering Performance After KL_NMF Dimensionality Reduction:
{'adjusted_rand_score': 0.860653095682575,
 'balanced_accuracy_score': 0.19504165619075656,
 'completeness_score': 0.8328000978296903,
 'homogeneity_score': 0.829462977945581,
 'mutual_info_score': 1.329140322141225,
 'v_measure_score': 0.8311281881263124}
```

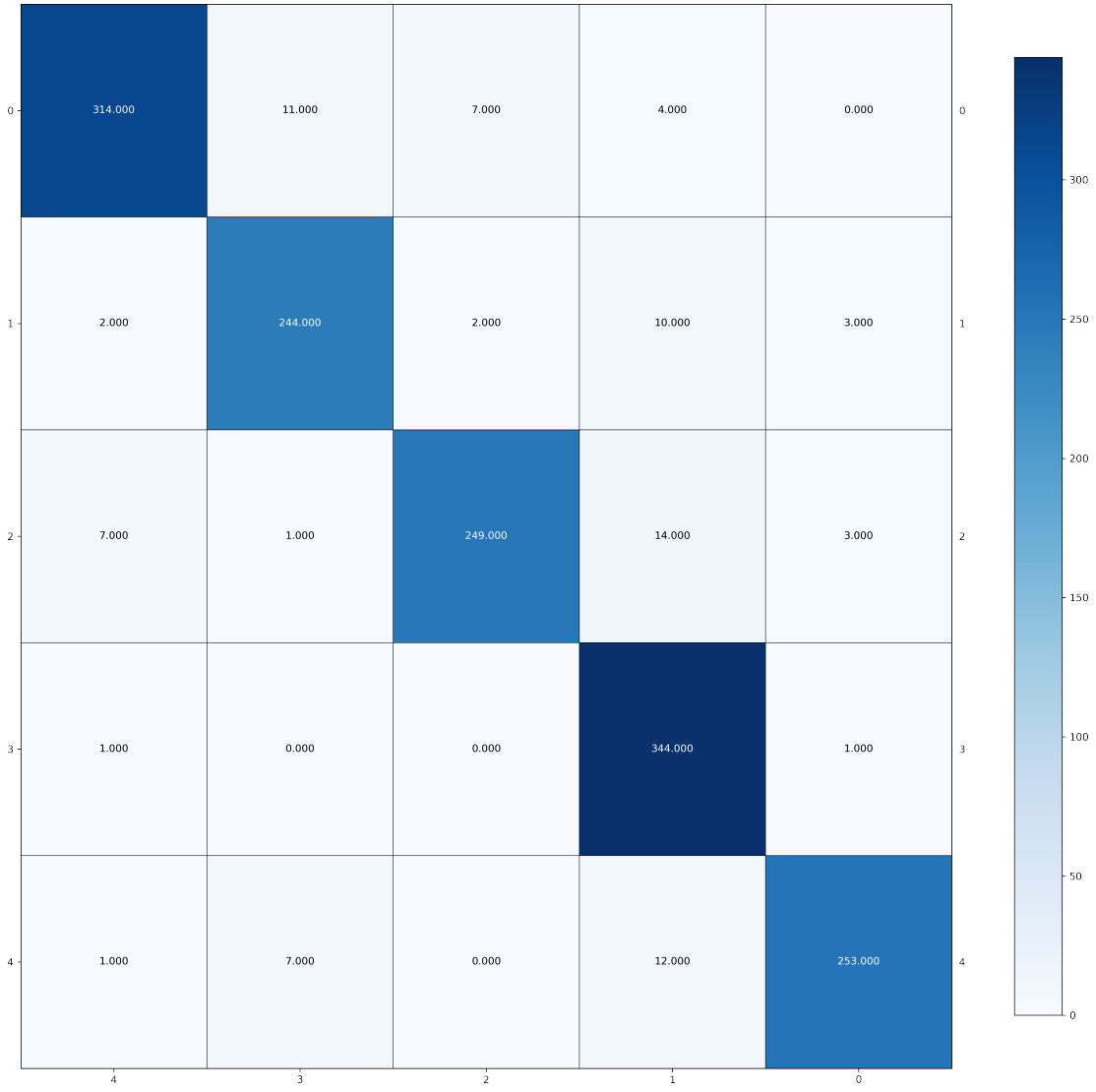


Figure 25: Contingency matrix of K-Means Algorithm after using KL-NMF with n_components=5.

Appendix: Source Code

Leave this commented out until we're ready to turn it in, since it slows down compilation

project2.py

```
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from pprint import pprint
from itertools import product
import json

from plotmat import plot_mat
from helpers import get_tf_idf,get_contingency,get_all_data,get_KMeans,get_cluster_metrics, LSI_builtin, get_NMF,
    close_factors,get_binary_dataset,get_LSI_matrix

np.random.seed(42)
random.seed(42)



def q1():
  twenty_all = get_all_data()
  X_tfidf = get_tf_idf(twenty_all.data)
  print(f"TF-IDF Matrix Shape (all data): {X_tfidf.shape}")


def q2():
  twenty_all = get_all_data()
  X_tfidf = get_tf_idf(twenty_all.data)
  n_clusters = 2
  predictions,kmeans = get_KMeans(X_tfidf,n_clusters=n_clusters)
  A = get_contingency(twenty_all.target,predictions)
  plot_mat(A,title=f"Contingency Table, K-Means",pic_fname="q2.pdf",sizemult=2.5)

def q3():
  twenty_all = get_all_data()
  X_tfidf = get_tf_idf(twenty_all.data)
  n_clusters = 2
  predictions,kmeans = get_KMeans(X_tfidf,n_clusters=n_clusters)
  pprint(get_cluster_metrics(twenty_all.target,predictions))

def q4():
  twenty_all = get_all_data()
  X_tfidf = get_tf_idf(twenty_all.data)
  rmax=1000
  X_lsi,svd = LSI_builtin(X_tfidf,rmax)
  expl_variances = { r : 100*sum(svd.explained_variance_ratio_[0:r]) for r in range(1,rmax+1)}
  fig,ax = plt.subplots()
  ax.plot(tuple(expl_variances.keys()), tuple(expl_variances.values()))
  ax.set_xlabel("Number of Principal Components $r$")
  ax.set_ylabel("Percent of Explained Variance in First $r$ Principal Components")
  ax.set_title("Percent Variance Explained by Top $r$ Principal Components vs. $r$")
  plt.tight_layout()
  plt.savefig("figures/q4_variance_explained_vs_pca_dimension.pdf")
```

```
  plt.close()

def q5():
  twenty_all = get_all_data()
  X_tfidf = get_tf_idf(twenty_all.data)
  rmax = 1000
  n_clusters = 2
  X_lsi = get_LSI_matrix(X_tfidf,rmax)
  rvals =  (1, 2, 3, 5, 10, 20, 50, 100, 300)
  metrics_lsi, metrics_nmf = [], []
  for r in rvals:
    predictions_lsi,kmeans_lsi = get_KMeans(X_lsi[:,:(r+1)],n_clusters=n_clusters)
    metrics_lsi.append(get_cluster_metrics(twenty_all.target,predictions_lsi))

    predictions_NMF,kmeans_nmf = get_KMeans(get_NMF(X_tfidf,r), n_clusters=n_clusters)
    metrics_nmf.append(get_cluster_metrics(twenty_all.target,predictions_NMF))

  df_lsi = pd.DataFrame(metrics_lsi)
  df_nmf = pd.DataFrame(metrics_nmf)
  columns = df_nmf.columns
  # for df in (df_lsi,df_nmf):
  # df['Dimension'] = rvals
  nrows,ncols = close_factors(len(columns))
  fig,axs = plt.subplots(nrows=nrows,ncols=ncols,sharex=True)
  for (col,ax) in zip(columns,fig.axes):
    for method,df in {"LSI": df_lsi,"NMF" : df_nmf}.items():
      ax.plot(rvals,df[col],label=method)
      ax.set_title(f"{col} vs.\nReduced Dimension $r$",fontsize=8)
      ax.set_ylabel(f"{col}", fontsize=6)
      ax.legend()
      plt.xticks(fontsize=8)
      plt.yticks(fontsize=8)
  # plt.xlabel("Reduced Dimension £r£")
  fig.add_subplot(111, frameon=False)
  plt.tick_params(labelcolor='none', top=False, bottom=False, left=False, right=False)
  plt.xlabel("Reduced Dimension $r$")
  plt.tight_layout() #plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
  plt.savefig("figures/q5_purity_vs_dimension.pdf")
  plt.close()


def q7(binary_dataset=False):
  data= get_binary_dataset() if binary_dataset else get_all_data()
  n_classes = len(set(data.target))
  n_samples = len(data.target)
  filename_base=f"figures/q7_kmeans_labels_predictions_2d_{n_classes}_classes"

  target = data.target #ground truth
  X_tfidf = get_tf_idf(data.data)
  # dimensionality reduction
  r=20
  n_clusters=2
  fig,axs = plt.subplots(nrows=2,ncols=2,sharex=True,sharey=True)
  performance_outfile = open(f"{filename_base}_performance.txt", "w")
  performance_outfile.write(f"{n_samples} Samples, {n_classes} Classes, {n_clusters} Clusters\n")
  for (method,transformer),axlist in zip({"LSI": get_LSI_matrix, "NMF" : get_NMF}.items(),axs):
    X_reduced = transformer(X_tfidf,r)
    predictions, _ = get_KMeans(X_reduced,n_clusters=n_clusters)
    X_2d = get_LSI_matrix(X_reduced,2)
    labels = {" Predictions" : predictions, "Ground Truth": target}
```

```
    performance_outfile.write(f"\nClustering Performance After {method} Dimensionality Reduction:\n")
    pprint(get_cluster_metrics(target,predictions),performance_outfile)
    for ax,(label_type,label) in zip(axlist,labels.items()):
      ax.scatter(
        X_2d[:,0],X_2d[:,1],
        c=label,
        s=1,
        alpha = 0.5,
        linewidths=0,
        edgecolors=None
      )
      ax.set_title(f"K-Means {label_type} Using {method}\n{n_classes} Classes, $\\in R^2$ via SVD",fontsize=8)
      ax.set_xticks([])
      ax.set_yticks([])
  plt.tight_layout() #plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
  plt.savefig(f"{filename_base}.png", dpi=350, transparent=False)
  plt.savefig(f"{filename_base}.pdf", transparent=False)
  plt.close()
  performance_outfile.close()

if __name__=="__main__":
  # for fn in (q1,q2,q3,q4,q5,q7):
  #   print(fn.__name__)
  #   fn()
  q7(binary_dataset=False)
```

helpers.py

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF

from caching import cached



def get_all_data(categories = None):
  return fetch_20newsgroups(categories=categories,shuffle=False,random_state=None,remove=('headers','footers'))



def get_binary_dataset(cats1=["comp.graphics", "comp.os.ms-windows.misc", "comp.sys.ibm.pc.hardware",
    "comp.sys.mac.hardware"],cats0=["rec.autos", "rec.motorcycles", "rec.sport.baseball", "rec.sport.hockey"]):
  data = fetch_20newsgroups(categories = cats1 + cats0,shuffle=False,random_state=None,remove=('headers','footers'))
  cat1_inds = [data.target_names.index(cat) for cat in cats1]
  data.target = [1 if target_ind in cat1_inds else 0 for target_ind in data.target]
  return data


def get_vectorizer(basic=False):
  return CountVectorizer(
    strip_accents='unicode',
    stop_words='english',
```

```
    ngram_range=(1,1), #default (1,1)
    token_pattern=r"(?u)\b(\d*[a-zA-Z]+\d*)+\b", #default r"(?u)\b\w\w+\b"
    min_df=3
  )

@cached
def get_tf_idf(train_data):
  vectorizer = get_vectorizer(basic=True) #no lemmatization
  X_train_counts = vectorizer.fit_transform(train_data)
  tfidf_transformer = TfidfTransformer()
  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
  return X_train_tfidf


def get_contingency(ground_truth_target,predictions):
  A = np.zeros((len(set(ground_truth_target)),len(set(predictions))))
  for (i,j) in zip(ground_truth_target,predictions):
    A[i,j] += 1
  return A

@cached
def get_KMeans(X,n_clusters=8):
  kmeans = KMeans(
    n_clusters=n_clusters,
    random_state=0,
    max_iter = 2000,
    n_init = 50
  )
  predictions = kmeans.fit_predict(X)
  return predictions,kmeans


def get_cluster_metrics(labels,predictions):
  metric_fns = (
    metrics.homogeneity_score,
    metrics.completeness_score,
    metrics.v_measure_score,
    metrics.adjusted_rand_score,
    metrics.mutual_info_score,
    metrics.balanced_accuracy_score
  )
  return { fn.__name__ : fn(labels,predictions) for fn in metric_fns}


@cached
def LSI_builtin(X,n_components):
  svd = TruncatedSVD(n_components=n_components)
  X_reduced= svd.fit_transform(X)
  return X_reduced,svd

def get_LSI_matrix(*args,**kwargs):
  X_reduced,svd = LSI_builtin(*args,**kwargs)
  return X_reduced

def ss_error(mat1,mat2):
  return np.linalg.norm(mat1-mat2,ord='fro')**2

@cached
def get_NMF(X,n_components):
  model = NMF(n_components=n_components)
```

```
  W = model.fit_transform(X)
  return W


def close_factors(n):
  return min([(n//i,i,abs((n//i) - i)) for i in range(1, int(n**0.5)+1) if n % i == 0],key=lambda tup: tup[2])[0:2]
```

caching.py

```
import os
from os.path import isfile,isdir
from glob import glob
import pickle
from functools import wraps
# from hashlib import sha1
from deepdiff import DeepHash
import time
# from copy import copy, deepcopy
# from mmh3 import hash128



printing = False
def log(*args,**kwargs):
  if printing:
    print(*args,**kwargs)


def timed(f):
  @wraps(f)
  def wrapped(*args,**kwargs):
    start=time.time()
    output = f(*args,**kwargs)
    end=time.time()
    log(f"{f.__name__} took {int(end-start)}s")
    return output
  return wrapped

def get_cachedir():
  return './cache'

def ensure_cache():
  cachedir = get_cachedir()
  if not isdir(cachedir):
    os.mkdir(cachedir)

@timed
def load(fname):
  log(f"loading {fname} (already executed)")
  with open(f"{fname}",'rb') as f:
    return pickle.load(f)

@timed
def dump(obj,fname):
  with open(f"{fname}",'wb') as f:
    pickle.dump(obj,f)

def pre_hash(obj):
```

```
  # numpy sparse array
  if hasattr(obj,'todok'):
    # return dict(obj.todok())
    canonical = obj.sorted_indices() #copies
    canonical.sum_duplicates()
    return (canonical.shape,tuple(canonical.indices),tuple(canonical.data))
  else:
    return obj

@timed
def get_hash(args,kwargs):
  hashable_args = [pre_hash(arg) for arg in args] #
  hashable_kwargs = {k : pre_hash(v) for k,v in kwargs.items()}
  obj = (hashable_args,hashable_kwargs)
  return DeepHash(obj,ignore_string_case=True,significant_digits=1)[obj]

@timed
def cache_execute(func,file_out_base,args,kwargs,hashed_args):
  log(f"EXECUTING (and caching) {func.__name__}")
  output = func(*args,**kwargs)
  dump(output,f"{file_out_base}{hashed_args}")
  return output

def cached(func):
  file_base = f"{get_cachedir()}/{func.__name__}"
  @wraps(func)
  def decorated(*args,**kwargs):
    ensure_cache()
    hashvals = [filename.split(file_base,1)[-1] for filename in glob(f"{file_base}*")]
    hashed_args = get_hash(args,kwargs)
    if hashed_args in hashvals:
      file_out = f"{file_base}{hashed_args}"
      return load(file_out)
    return cache_execute(func,file_base,args,kwargs,hashed_args)
  return decorated
```

plotmat.py

```
import itertools
import os
from os.path import isfile,isdir
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

def get_figdir():
    return './figures'


def ensure_figdir(fname):
    figdir = get_figdir()
    if not isdir(figdir):
        os.mkdir(figdir)
    if fname.startswith(figdir) or fname.startswith(figdir.strip('.')) or fname.startswith(figdir.strip('./')):
        return fname
    else:
        return f"{figdir}/{fname}"
```

```
def plot_mat(mat, xticklabels = None, yticklabels = None, pic_fname = None, size=(-1,-1), if_show_values = True,
             colorbar = True, grid = 'k', xlabel = None, ylabel = None, title = None, vmin=None, vmax=None,sizemult=1):
    if size == (-1, -1):
        size = (sizemult*mat.shape[1], sizemult*mat.shape[0])

    fig = plt.figure(figsize=size)
    ax = fig.add_subplot(1,1,1)

    # im = ax.imshow(mat, cmap=plt.cm.Blues)
    im = ax.pcolor(mat, cmap=plt.cm.Blues, linestyle='-', linewidth=0.5, edgecolor=grid, vmin=vmin, vmax=vmax)

    if colorbar:
        plt.colorbar(im,fraction=0.046, pad=0.06)
    # tick_marks = np.arange(len(classes))
    # Ticks
    lda_num_topics = mat.shape[0]
    nmf_num_topics = mat.shape[1]
    yticks = np.arange(lda_num_topics)
    xticks = np.arange(nmf_num_topics)
    ax.set_xticks(xticks + 0.5)
    ax.set_yticks(yticks + 0.5)
    if xticklabels is None:
        xticklabels = [str(i) for i in xticks]
    if yticklabels is None:
        yticklabels = [str(i) for i in yticks]
    ax.set_xticklabels(xticklabels)
    ax.set_yticklabels(yticklabels)

    # Minor ticks
    # ax.set_xticks(xticks, minor=True);
    # ax.set_yticks(yticks, minor=True);
    # ax.set_xticklabels([], minor=True)
    # ax.set_yticklabels([], minor=True)

    # ax.grid(which='minor', color='k', linestyle='-', linewidth=0.5)

    # tick labels on all four sides
    ax.tick_params(labelright = True, labeltop = False)

    if ylabel:
        plt.ylabel(ylabel, fontsize=15)
    if xlabel:
        plt.xlabel(xlabel, fontsize=15)
    if title:
        plt.title(title, fontsize=15)

    # im = ax.imshow(mat, interpolation='nearest', cmap=plt.cm.Blues)
    ax.invert_yaxis()

    # thresh = mat.max() / 2
    def show_values(pc, fmt="%.3f", **kw):
        pc.update_scalarmappable()
        ax = pc.axes
        for p, color, value in itertools.zip_longest(pc.get_paths(), pc.get_facecolors(), pc.get_array()):
            x, y = p.vertices[:-2, :].mean(0)
            if np.all(color[:3] > 0.5):
                color = (0.0, 0.0, 0.0)
            else:
                color = (1.0, 1.0, 1.0)
            ax.text(x, y, fmt % value, ha="center", va="center", color=color, **kw, fontsize=10)
```

```
    if if_show_values:
        show_values(im)
    # for i, j in itertools.product(range(mat.shape[0]), range(mat.shape[1])):
    #     ax.text(j, i, "{:.2f}".format(mat[i, j]), fontsize = 4,
    #             horizontalalignment="center",
    #             color="white" if mat[i, j] > thresh else "black")
    plt.tight_layout()
    if pic_fname:
        plt.savefig(ensure_figdir(pic_fname), dpi=300, transparent=True)

    # plt.show()
    plt.close()
```

References

[1] Aude Genevay, Gabriel Dulac-Arnold, and Jean-Philippe Vert. Differentiable deep clustering with cluster size constraints, 2019.

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[3] Divya G Soman KP Rakesh Peter, Shivapratap Gopakumar. Evaluation of svd and nmf methods for latent semantic analysis. In *International Journal of Recent Trends in Engineering*, May 2009.

[4] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics.