Data Mining Assignment 3
Clustering

Maksim Karnaukh
1st year Masters at Department of Computer Science
University of Antwerp
Antwerp, Belgium
Email: maksim.karnaukh@student.uantwerpen.be

1. Introduction

In this report we will work with a dataset consisting of news articles. The goal for us is to divide these articles into meaningful clusters, experimenting with different data preprocessing techniques, clustering algorithms and distance functions.

2. Data Preprocessing

Since the news articles consist of texts, we must preprocess them to make them understandable to a computer, basically transforming the text into numbers. Before we do this vectorization, we will first perform some processing steps on the text itself.

2.1. Text Preprocessing Pipeline

To do this, we create a pipeline that tranforms our text, according to the following steps:

1) **Lowercasing**: Convert all text to lowercase.
2) **Punctuation Removal**: Eliminate punctuation marks from the text.
3) **Tokenization**: Split the text into individual words or tokens for further analysis.
4) **Stop Word Removal**: Filter out common words (stop words) that do not carry significant meaning in the context.
5) **Number Removal**: Exclude numerical digits from the text to simplify analysis and focus on textual content.
6) **Specific Token Removal**: Remove specific tokens that may not be relevant to the analysis.
7) **Stemming**: Reduce words to their root or base form to normalize variations of words.
8) **Unicode Character Removal**: Eliminate any remaining non-ASCII characters to ensure compatibility and consistency in the text.

The above steps adhere more or less to the standard method of preprocessing text for vectorization. The specific token removal is something I manually did in order to remove words such as 'advertisement', which don't have anything to do with the article itself in this case. Another thing we do, inspired by [3], is removing the top 1% of the most frequent and top 1% of the least frequent words.

Our dataset has three text columns for each article: 'headlines', 'description' and 'content'. We concatenate the text of the three columns into one big text and then apply the preprocessing function to it.

2.2. Text Vectorization

After preprocessing the text data, the next step is to convert the preprocessed text into numerical vectors that can be better understood by our (clustering) algorithms. One of the simplest methods to do so is the "bag of words" approach. However, one widely used method that has good advantages over the "bag of words" approach for this purpose is TF-IDF (Term Frequency-Inverse Document Frequency). Below is the explanation, although this algorithm is a pretty standard one.

TF-IDF assigns weights to each term in a document based on its frequency in the document (TF) and its rarity across all documents (IDF). This helps in capturing the importance of terms in individual documents while also considering their significance in the entire corpus.

The TF-IDF vectorization process involves the following steps:

1) **Term Frequency (TF)**: Calculate the frequency of each term in the document.
2) **Inverse Document Frequency (IDF)**: Compute the inverse document frequency for each term, which measures how rare a term is across all documents.
3) **TF-IDF Calculation**: Multiply the TF and IDF values for each term to obtain the TF-IDF weight.

The final result that we get here is a vector of TF-IDF weights per article. To reduce the amount of dimensions of our dataset, I also removed all features with a document frequency of less than 5 using the min_df parameter of the TF-IDF vectorizer.

3. Clustering

For clustering, selecting a suitable distance function for the clustering algorithm(s) is considered the most important factor. We will discuss K-means, K-medoids, Bisecting K-means (distance-based partitional clustering algorithms) and Agglomerative clustering (hierarchical clustering algorithm). We will not discuss DBSCAN in detail, because we have very high dimensional data and it only works well when the density is uniform. DBSCAN doesn't give good results in our case.

3.1. Cluster Validation

Something that is quite difficult to check, but is necessary for cluster analysis is cluster validity. We will mainly use three cluster validation methods [2] to measure how good a clustering algorithm result is: the correlation of the incidence and proximity matrices, the similarity matrix visualization and the use of a statistical framework.

The correlation is computed by first building the proximity matrix (which is basically the distance matrix) and the incidence matrix. The incidence matrix has one row and one column for each data point, where an entry is 1 if the associated pair of points belong to the same cluster and 0 otherwise. We then compute the correlation between these two matrices.

For the similarity matrix, we order the data points with respect to cluster labels and inspect the matrix visually using a heatmap.

The last one makes use of statistics to interpret our measures and provide a framework for cluster validity. The main idea here is to permutate the features of our dataset (only joint distribution is distorted) a certain number of times where each time we get a reference score (SSE or something else). In the end we can compute the mean and standard deviation of the list of scores for the randomly permuted data that we got and look at the Z-score to see how atypical our actual clustering result is compared to those random data results.

3.2. Clustering algorithms

Our dataset has 2500 news articles. The clusters we will generate will most likely represent the topics of the news articles, e.g. business, entertainment, etc. As per assignment, we will not generate more than 10 clusters. For clustering algorithms that require the number of clusters as input, we will make use of the elbow plot. This means that for each $k = 1, ..., 10$ amount of clusters, we run the clustering algorithm to determine the best score (e.g. for k-means we can use the SSE) and look for the 'elbow' on the plot where the score decreases proportionally less compared to before (for previous k values). We will now discuss all combinations of cluster algorithm, distance function and number of clusters (which we will generally denote as k) and highlight the interesting/good results.

3.2.1. K-means. For k-means, I used the sklearn cluster library. However, here we can't pass a distance metric as a parameter since sklearn's k-means uses the euclidean distance by default. Even if we couldn't use the cosine distance metric, since there is a linear relationship between euclidean and cosine distance for normalized vectors [1], and since the TF-IDF vectorizer uses L2-normalization, we could just use the euclidean distance.

Another option is to precompute a distance matrix (using cosine distance in this case), and use this for our k-means clustering with euclidean distance metric. The reason we focus on cosine distance so much is because we are working with text data. We have a large number of features (dimensions). Cosine similarity (which equals $1 - cosinedistance$) doesn't care about the magnitude of the vectors, and equals the cosine of the angle between the vectors.

We will first look at the elbow plots, both for euclidean and cosine distances. For these plots we used the KElbowVisualizer from the yellowbrick library. The cosine distances elbow plot annotates the elbow at $k = 4$, but 5 is a more reasonable choice there too. We will perform K-means for $k = 5$.

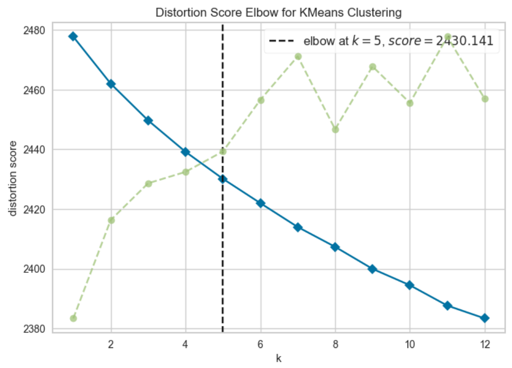


Figure 1: Elbow plot for k-means using euclidean distance on the normal TF-IDF matrix.

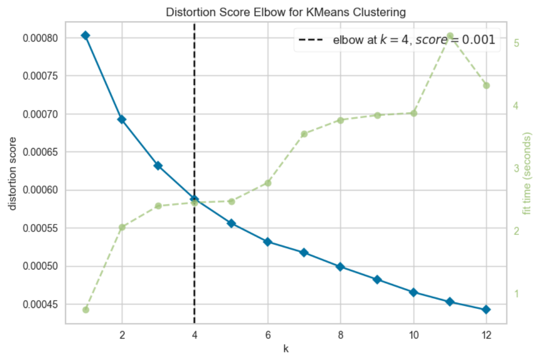


Figure 2: Elbow plot for k-means using the cosine distances matrix of the TF-IDF matrix.

When we say that we use the cosine distance matrix, it means we use a precomputed distance matrix using the cosine distance between data points. We can see the metrics in table 1. We will generally use the inertia, which in this case is the SSE. We will also try to look at the silhouette score (a measure of how close each point in one cluster is to points in the neighboring clusters) and the correlation for cluster validity. For each of these, we will also note the statistical reference mean and standard deviation for randomized data as we previously discussed. TF-IDF matrix means we only passed the TF-IDF matrix to the clustering algorithm. When we do dimensionality reduction (using SVD) on the TF-IDF matrix, we write the amount of dimensions of the resulting matrix that we then pass to the clustering algorithm.

Using the vectorized data or the cosine distance matrix of the vectorized data for the k-means algorithm yields alright results in terms of the evaluation scores and statistics, but the similarity matrices are not very good. The clusters here are very disproportional, with one cluster containing a lot of points for both combinations.

The rest also have decent results, with the k-means clustering on the precomputed cosine distance matrix of the SVD reduction to 20 dimensions having the best resulting similarity matrix. The amount of points per cluster here is also a little more balanced.

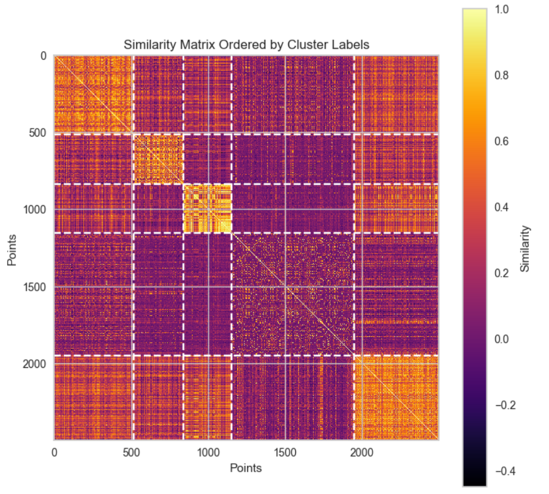


Figure 3: Similarity matrix of k-means clustering on the precomputed cosine distance matrix of the SVD reduction to 20 dimensions

3.2.2. Bisecting K-means.

Bisecting k-means follows the same approach in terms of explanation. Looking at the elbow plots, the best value for k is in the range 4 to 6. We will choose 5 here too. Just like with k-means, the statistics show that the clusters are indeed valid. The best result is the one with the same combination as k-means, meaning bisecting k-means clustering on the precomputed cosine distance matrix of the SVD reduction to 20 dimensions.

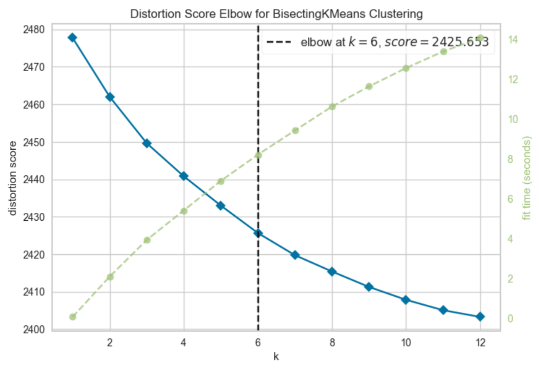


Figure 4: Elbow plot for k-means using euclidean distance on the normal TF-IDF matrix.

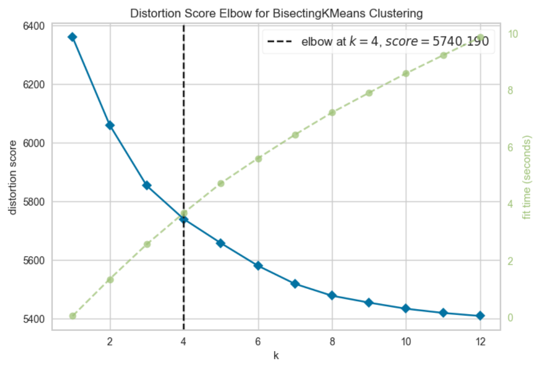


Figure 5: Elbow plot for k-means using the cosine distances matrix of the TF-IDF matrix.

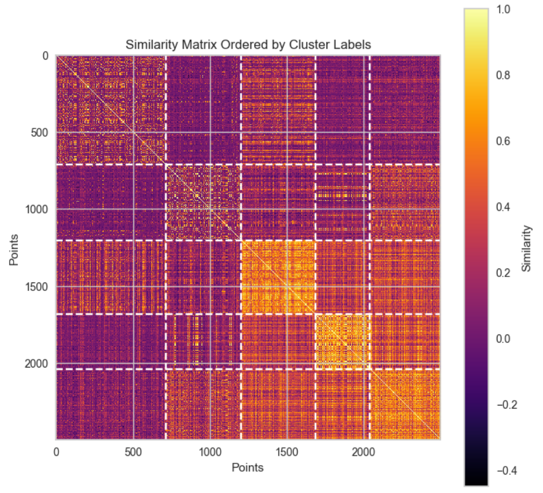
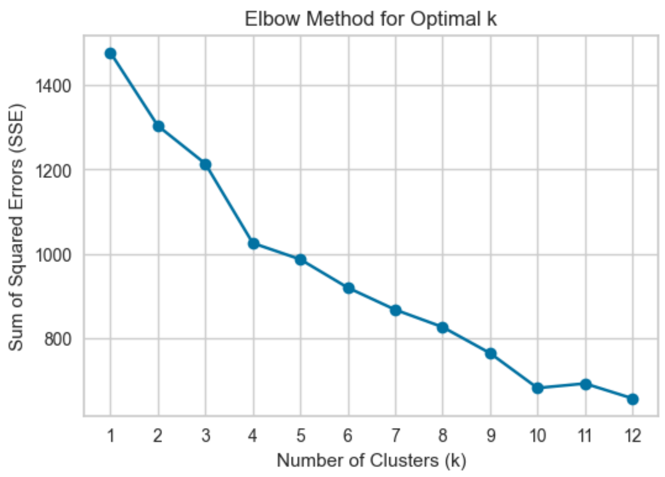


Figure 6: Similarity matrix of bisecting k-means clustering on the precomputed cosine distance matrix of the SVD reduction to 20 dimensions

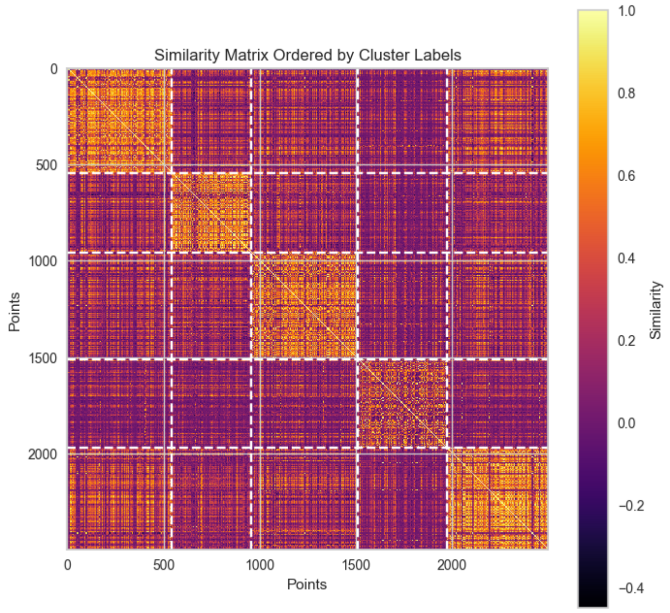
3.2.3. K-Medoids.

For sklearn's k-medoids, it is possible to provide a distance metric. Here we will also try to employ dimensionality reduction, since this gives better results.

Since it is not clear from the elbow plot, we will try to

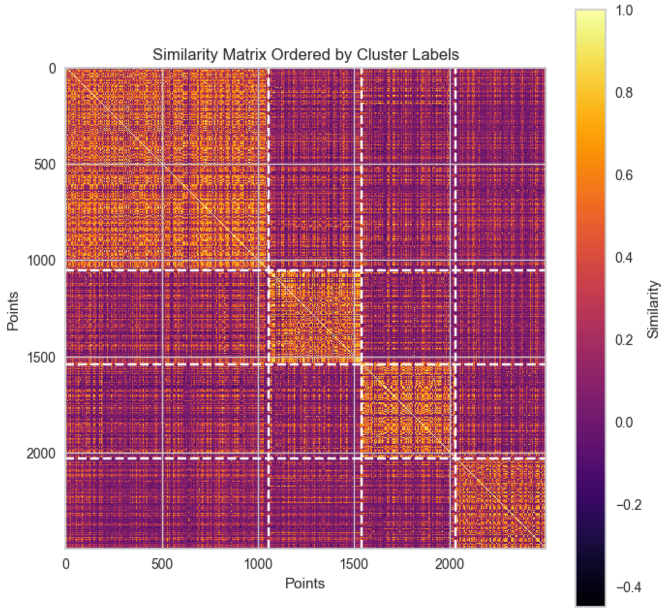


(a)

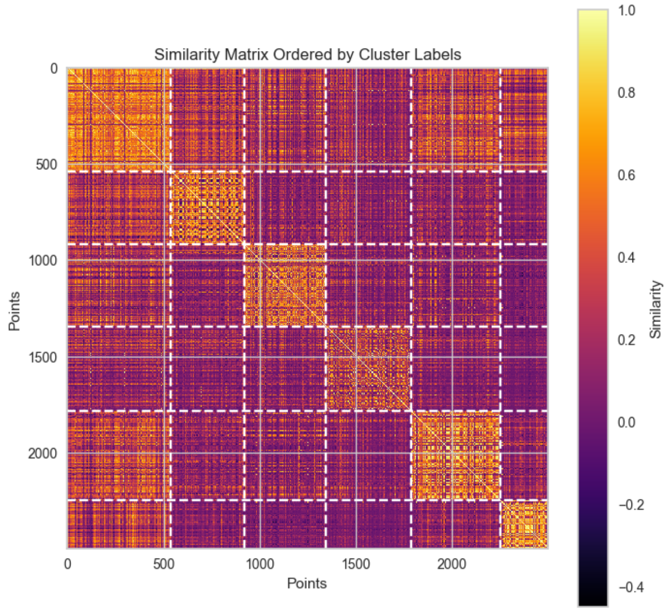


(b)

Figure 7: (Left) Elbow plot for k-medoids using cosine distance metric and the SVD reduction to 20 dimensions. (Right) Similarity matrix of k-medoids clustering using cosine distance metric and the SVD reduction to 20 dimensions for k=5.



(a)



(b)

Figure 8: (Left) Similarity matrix of k-medoids clustering using cosine distance metric and the SVD reduction to 20 dimensions for k=4. (Right) Similarity matrix of k-medoids clustering using cosine distance metric and the SVD reduction to 20 dimensions for k=6.

look at $k = 4, 5, 6$. All clustering results, as we can see in table 3, are valid in terms of reference statistics. It seems like the clustering with $k = 5$ is the best one, with a good similarity matrix. The amount of points per cluster are also well distributed.

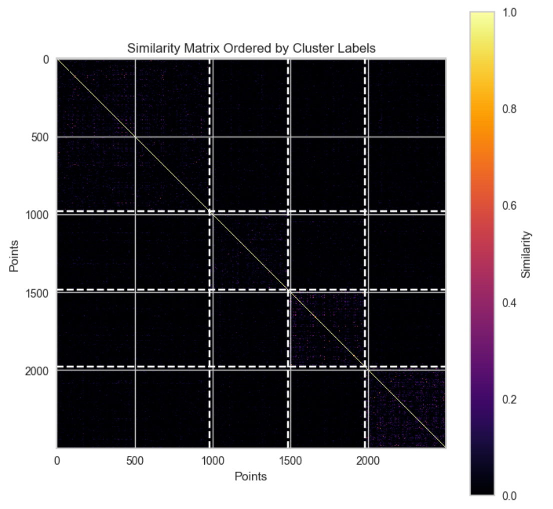


Figure 9: Similarity matrix of agglomerative clustering on the TF-IDF matrix using cosine distance metric and average linkage for k=4.

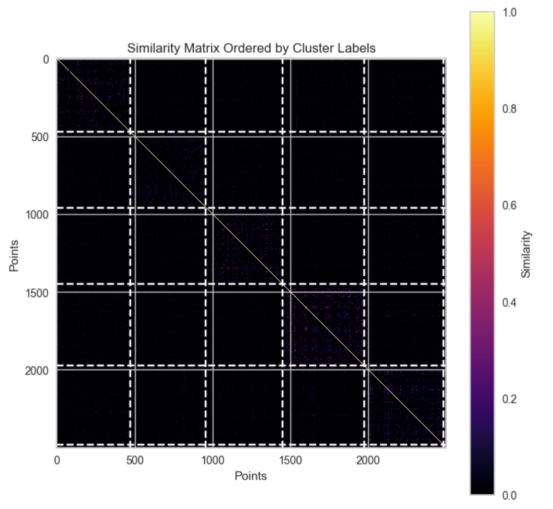


Figure 10: Similarity matrix of agglomerative clustering on the TF-IDF matrix using cosine distance metric and average linkage for k=6.

3.2.4. Agglomerative clustering. For agglomerative clustering, we will also look at $k = 4, 5, 6$. I have looked at different combinations, including linkage (average, complete, single) and distance metrics (cosine, jaccard). Everything except average linkage and cosine distance metric gave quite bad results. In table 4 we can see that the agglomerative clustering on the TF-IDF matrix using cosine distance and average linkage gives valid clustering for all considered k values.

It might not be clear to see from the similarity matrices since it is almost all black (0 similarity), but the clusters on the diagonal actually contain little dots of higher similarity if we zoom in a bit. The similarity matrix for k=6 contains

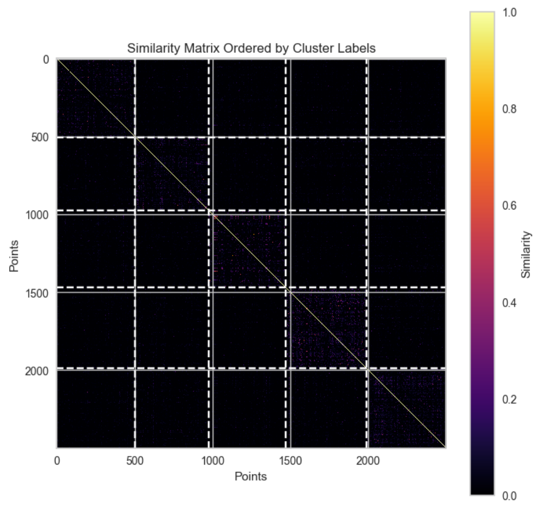


Figure 11: Similarity matrix of agglomerative clustering on the TF-IDF matrix using cosine distance metric and average linkage for k=5.

a small sixth cluster with only 17 data points, and looking at the big cluster of the k=4 similarity matrix, this makes me believe that k=5 gives the best result here. The amount of points per cluster here are very even (around 500 per cluster). This is also the reason why I sometimes talk about the distribution of data points per cluster. After inspecting the articles in each cluster, we could say that there are five different categories or topics of new articles: business, entertainment, sports, technology and education.

3.3. Discussion & Conclusion

The best results for k-means and bisecting k-means were using a precomputed cosine distance matrix of the 20 dimensions SVD reduced TF-IDF matrix for k=5. K-medoids also performed well for k=5 and using the cosine distance metric for the TF-IDF matrix reduced to 20 dimensions, together with agglomerative clustering using cosine distance and average linkage on the TF-IDF matrix.

Some results were not discussed here because they gave very bad results, e.g. jaccard distance with k-medoids. It's entirely possible that the reason why the clusterings discussed in this paper and the ones that gave worse results are either bad or not optimal simply because of the way we preprocessed our data.

One thing I would like to discuss quickly is the fact that my preprocessing pipeline removes digits and numbers from the texts. I thought it could be a good idea to count the amount of integers and the amount of floats in the text before removing them and then adding these two features to my processed dataframe. The idea behind this is that articles that are related to economics, business and financial topics would have more numbers, and more specifically floats in their texts. Sports articles would normally also contain more integer numbers than the other topics. However, when

I implemented this little addition, and after normalizing these features, it turned out to give worse results on almost everything I had up to that point.

4. Notes

The code and plots/pictures for this project can be found at https://github.com/MaksimKarnaukh/DataMining. In the assignment3/output/cluster_assignments/ folder, you can find the clusters.xlsx file with the original data together with the assigned cluster labels.

References

[1] Ajay Patel. Relationship between Cosine Similarity and Euclidean Distance, 2020. https://ajayp.app/posts/2020/05/relationship-between-cosine-similarity-and-euclidean-distance/.

[2] Huzefa Rangwala. Clustering II, 2016. https://piazza.com/class_profile/get_resource/ircffd3731a4n0/iu5o8ptzxdz5ev.

[3] Siddhant Sadangi. Introduction to Text Classification in Python, 2019. https://medium.com/analytics-vidhya/introduction-to-text-classification-in-python-659eccf6b2e.

TABLE 1: Results for K-means clustering.

K-means	**Inertia**	**Inertia** (μ, σ)	**Silhouette**	**Silhouette** (μ, σ)	**Corr.**	**Corr.** (μ, σ)
TF-IDF matrix	2430	(2459, 2.36)	0.0066	(0.017, 0.0084)	-0.0766	(-0.0042, 0.0044)
20 dimensions (SVD) of TF-IDF matrix	138.7	(153.2, 0.25)	0.3124	(0.12, 0.0049)	-0.4273	(-0.343, 0.0067)
5 dimensions (SVD) of TF-IDF matrix	21.2339	(33.34, 0.191)	0.5532	(0.3517, 0.0051)	-0.6941	(-0.6338, 0.006)
cos. dist. of 20 dimensions (SVD) of TF-IDF matrix	168591	(297409, 7.43)	0.201	(0.0001, 0.0001)	-0.5712	(-0.0215, 0.0044)
sq. eucl. dist. of 20 dimensions (SVD) of TF-IDF matrix	10381	(72707, 2.69)	0.44	(-0.0001, 0.0003)	-0.3992	(-0.0185, 0.0060)
cos. dist. of TF-IDF matrix	5591	(6335, 1.81)	0.10	(-0.008, 0.0108)	-0.6193	(-0.08, 0.04)

TABLE 2: Results for Bisecting K-means clustering.

Bisecting K-means	**Inertia**	**Inertia** (μ, σ)	**Silhouette**	**Silhouette** (μ, σ)	**Corr.**	**Corr.** (μ, σ)
TF-IDF matrix	2433	(2453, 1.08)	0.0074	(0.0091, 0.0037)	-0.1209	(-0.0053, 0.0066)
20 dimensions (SVD) of TF-IDF matrix	141.6945	(156.5, 1.28)	0.2186	(0.06, 0.0285)	-0.4448	(-0.31, 0.01)
5 dimensions (SVD) of TF-IDF matrix	25.28	(38.14, 1.24)	0.4839	(0.1899, 0.0477)	-0.7036	(-0.4294, 0.07)
cos. dist. of 20 dimensions (SVD) of TF-IDF matrix	178931	(297383, 8.18)	0.15	(0.0001, 0.0001)	-0.514	(-0.021, 0.003)
cos. dist. of TF-IDF matrix	5658	(6331, 2.08)	0.028	(0.0127, 0.0108)	-0.4727	(-0.11, 0.037)

TABLE 3: Results for K-medoids clustering.

K-medoids	**Inertia**	**Inertia** (μ, σ)	**Silhouette**	**Silhouette** (μ, σ)	**Corr.**	**Corr.** (μ, σ)
TF-IDF matrix, cosine	2343	(2397, 5.76)	0.0032	(-0.0033, 0.0015)	-0.107	(-0.047, 0.0085)
20 dimensions (SVD) of TF-IDF matrix, cosine, k=5	933	(1335, 26.88)	0.058	(0.0268, 0.0085)	-0.46	(-0.255, 0.02)
20 dimensions (SVD) of TF-IDF matrix, cosine, k=4	1059	(1378, 46)	0.066	(0.024, 0.011)	-0.434	(-0.236, 0.025)
20 dimensions (SVD) of TF-IDF matrix, cosine, k=6	909	(1284, 43)	-0.025	(0.027, 0.0076)	-0.44	(-0.266, 0.03)

TABLE 4: Results for agglomerative clustering.

Agglomerative clustering	**Silhouette**	**Silhouette** (μ, σ)	**Corr.**	**Corr.** (μ, σ)
TF-IDF matrix, cosine, average linkage, k=4	0.0058	(-0.0183, 0.02)	-0.205	(-0.022, 0.003)
TF-IDF matrix, cosine, average linkage, k=5	0.007	(-0.012, 0.016)	-0.246	(-0.025, 0.003)
TF-IDF matrix, cosine, average linkage, k=6	0.006	(-0.03, 0.03)	-0.2476	(-0.027, 0.002)
15 dimensions (SVD) of TF-IDF matrix, cosine, average linkage, k=5	-0.025	(0.027, 0.0076)	-0.44	(-0.266, 0.03)