EE219 Project 2

Clustering Analysis

Winter 2018

Weijie Tang 305029285 twjeric@cs.ucla.edu

Wandi Cui 905024671 wandicui@ucla.edu

1. Introduction

In this project, we try to conduct clustering analysis on textual data and compare the results of using different transforms. Our work mainly includes following steps:

1) Feature Extraction and Selection (get TFxIDF, implement dimension reduction method such as LSI and NMF);
2) Apply K-means clustering algorithm to solving the problem;
3) Find top *r* principle component for each dimension reduction method by comparing different kinds of clustering metrics;
4) Explore the effects of different data preprocessing transformations on clustering performance;
5) Expand the dataset and repeat all the previous steps;
6) Data Visualization.

2. Construct Feature Vectors (problem 1)

We started with a well separable portion of the “20 Newsgroups” dataset, with eight labels. Then we can group them into two classes as shown in Table 2.1. We utilized the NLTK package to tokenize the text raw data and transformed the documents into TF-IDF vectors. With the parameters min_df = 3 and stop_words = ‘english’, we got 7882 documents with 15675 terms.

Table 2.1. Labels of two classes as ground truth

Class 1	'comp.graphics'	'comp.os.ms-windows.misc'	'comp.sys.ibm.pc.hardware'	'comp.sys.mac.hardware'
Class 2	'rec.autos'	'rec.motorcycles'	'rec.sport.baseball'	'rec.sport.hockey'

3. K-means Without Data Preprocessing (problem 2)

We pretended the labels are not available and aimed to find groupings of the documents so that documents with similar terms are in the same cluster. First, we applied K-means to original data without any preprocessing, then we evaluated the clustering results by several metrics: homogeneity score, the completeness score, the V-measure, the adjusted Rand score and the adjusted mutual info score, as well as the contingency table. For these metrics, we can find functions in Python package “sklearn.metrics”, and as for the contingency table, we realized it in the following way:

```
def contingency_table(true_labels, pre_labels):
    n_clusters = len(np.unique(pre_labels))
    A = np.zeros(shape = (n_clusters,n_clusters))
    uniq_true = list(set(true_labels))
    for i, true_label in enumerate(uniq_true):
        for j, pre_label in enumerate(pre_labels):
            if(true_labels[j] == true_label):
                A[i][pre_label] += 1
    return A
```

When setting k = 2, the results shows below:
Homogeneity: 0.398
Completeness: 0.440
V-measure: 0.418
Adjusted Rand-Index: 0.398
Adjusted Mutual Information Score: 0.398
Contingency Table:
[[1417 2486]
 [3942 37]]

4. K-means With Data Preprocessing

Since K-means cannot perform well when the data dimension is large, in this section, we applied two dimensionality reduction measure: LSI and NMF to preprocess the original data, and compared the results we got with previous results. Apart from this, we also used a few data normalization method to avoid the data scale difference influencing the K-means results.

4.1 Dimensionality Reduction (problem 3)

In this part, we utilized two methods to accomplish the dimensionality reduction. The original dimension of our data is 15675, so we tried to reduce it within 1000 and plotted the percent of variance the top r principal components can retain.

To complete this task, we didn't call function "svd" for 1000 times, but get the 1000-d matrix and remove the least important feature each time to construct the other dimensional matrix. The formula of the ratio is: $\frac{tr(X_r^T X_r)=\sum_{i=1}^{r} S_{ii}^2}{tr(X^T X)}$ What is worth to mention is that the TFIDF matrix is saved as a sparse matrix, so in order not to increase the storage capacity, we did not translated it back to whole matrix but used "diagonal().sum()" to get the trace of the matrix. The essential codes are shown below:

```
from sklearn.decomposition import TruncatedSVD
# Reduce dimension to 1000
svd = TruncatedSVD(n_components = 1000, n_iter = 7, random_state = 42)
lsiVectors = svd.fit_transform(tfidfVectors)
total_var = np.dot(tfidfVectors.T,tfidfVectors).diagonal().sum()
dim1000_diag = np.diag(np.dot(lsiVectors.T,lsiVectors))
var_retained = [0 for i in range(1000)]
var_retained[0] = dim1000_diag[0]
for i in range(1,1000):
    var_retained[i] = var_retained[i - 1] + dim1000_diag[i]
ratio_of_variance = var_retained / total_var
print(ratio_of_variance)
```

The figure 4.1 shows the plot result:

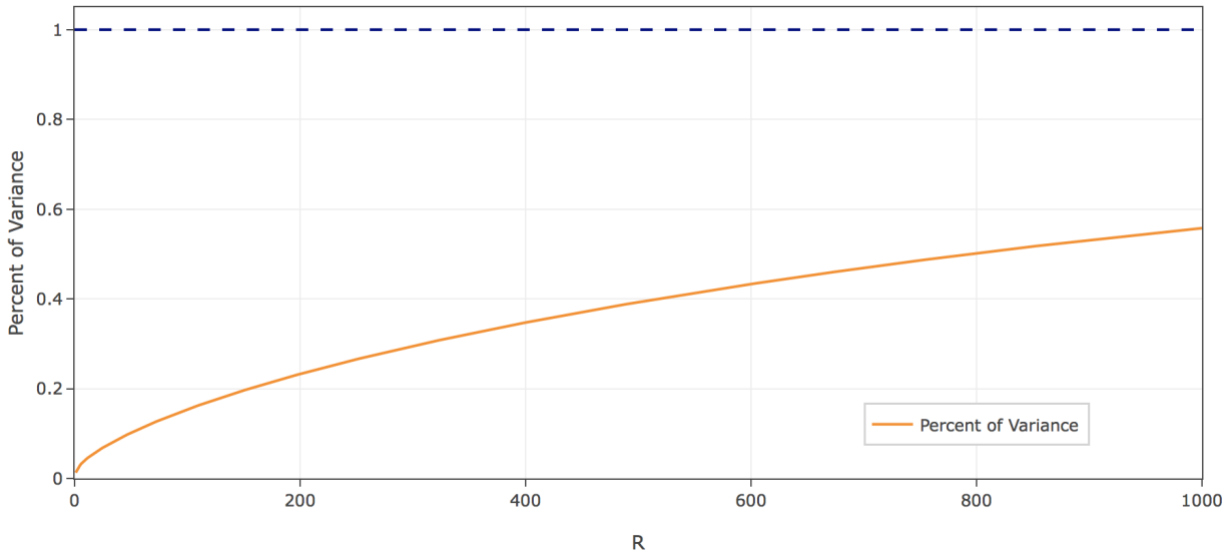


Figure 4.1 The plot of percent of variance retained by the top r principal components

Not surprisingly, the result is monotonically increasing, and 1000 dimensional matrix can only keep a slightly over 50% information of the original data.

Next, we tried specified values of r (1,2,3,5,10,20,50,100,300) to do the dimensionality reduction with both SVD and NMF. The figure 4.2 and 4.3 shows the result of K-means after SVD and NMF respectively in different values of r as well as the best value of r.

We can see that the best value of r for SVD is 3; and for NMF, the best value of r is 2.

No matter what dimensional reduction method we chose, the relation between dimension r and K-means results has a non-monotonic behavior. The reason is as follows: When r is small, the vector can only keep little information(features) of original data, so the clustering results won't be good; however when r is too big, which means the data are in a high-dimensional space, in this situation, the Euclidean distance is not a good metric anymore,

and K-means is an algorithm based on Euclidean distance, so the result will still not be good when r is big.

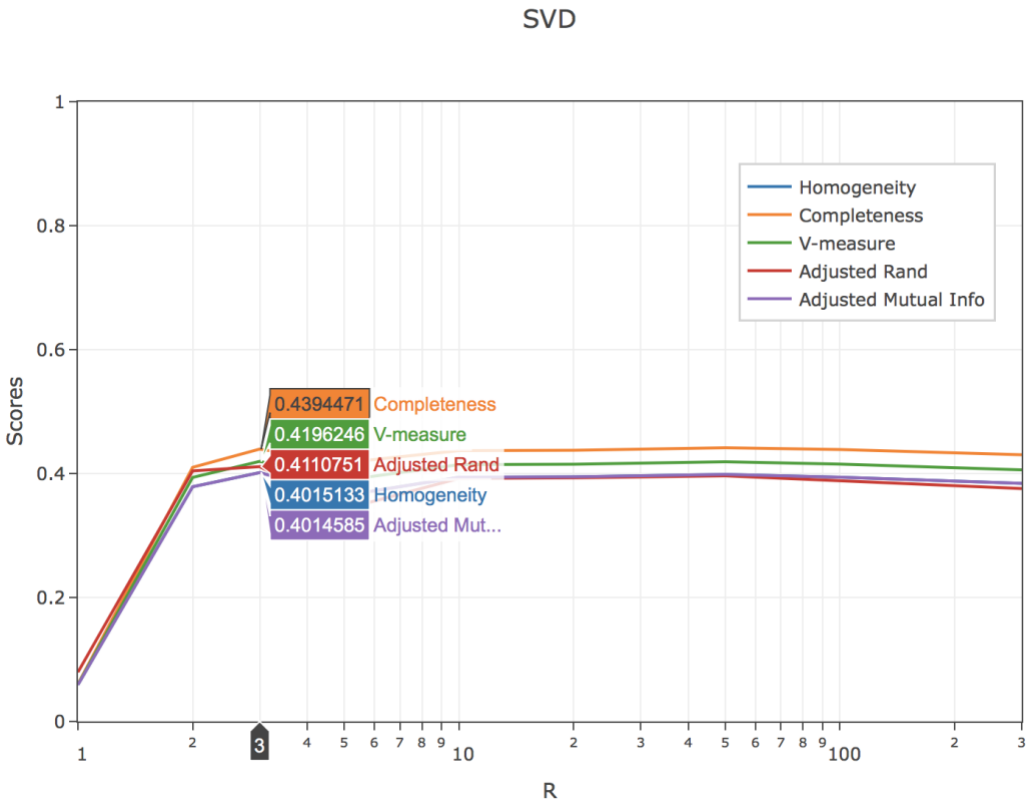


Figure 4.2 Clustering scores of different *r* for SVD (best r = 3)

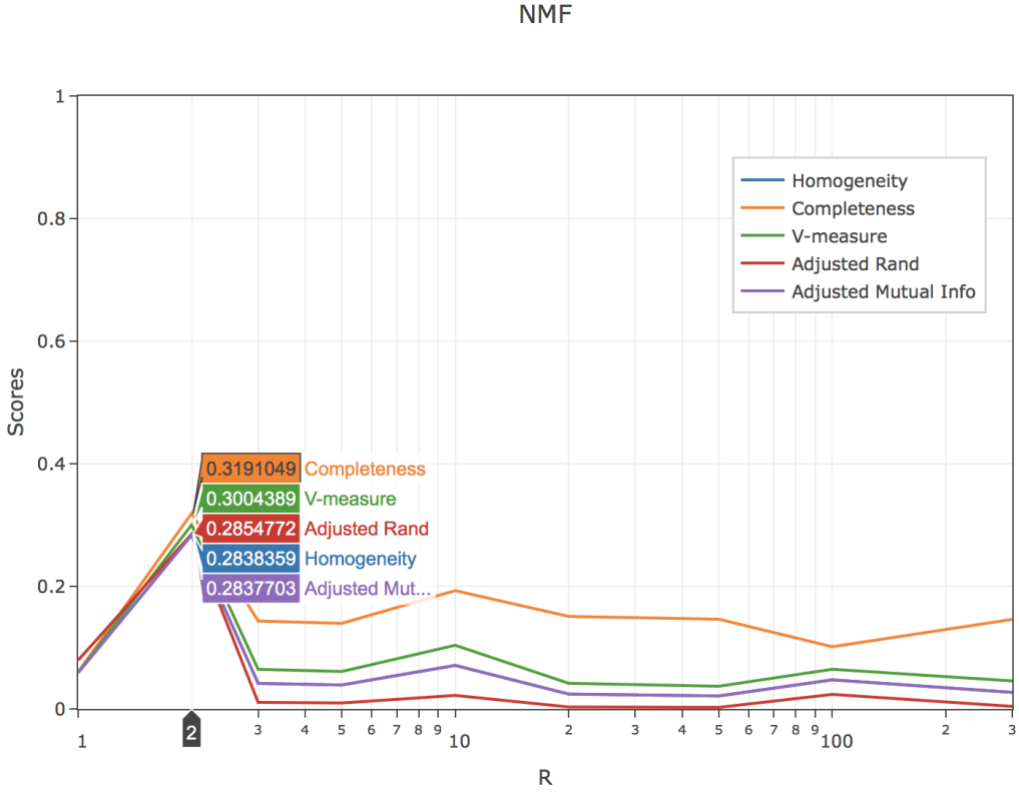


Figure 4.3 Clustering scores of different *r* for NMF (best r = 2)

4.2 Data Transformation (problem 4)

For SVD (best r = 3), we plotted the clustering results by projecting the data vectors onto 2 dimensional plane. Figure 4.4 shows the cluster results.

And the evaluation metrics are:

Best r for SVD: 3
Homogeneity: 0.401
Completeness: 0.439
V-measure: 0.419
Adjusted Rand-Index: 0.411
Adjusted Mutual Information Score: 0.401
Contingency Table:
[[2539 1364]
[51 3928]]

Figure 4.4 Cluster result for best SVD

For NMF (best r = 2), we plotted the clustering results by projecting the data vectors onto 2 dimensional plane. Figure 4.5 shows the cluster results.

And the evaluation metrics are:

Best r for NMF: 2
Homogeneity: 0.284
Completeness: 0.319
V-measure: 0.300
Adjusted Rand-Index: 0.285
Adjusted Mutual Information Score: 0.284
Contingency Table:
[[137 3766]
[2281 1698]]

Figure 4.5 Cluster result for best NMF

To improve the K-means results, we applied some data transformations, such as normalizing features into unit-variance features, logarithm transformation for NMF. The table 4.1 shows the results comparison of different groups of data transformation.

According to the table above, we could see that without any preprocessing, SVD could do better than NMF, and both of them get best result when r is a really small value. However, after data transformation, NMF gets better results than SVD, and SVD even gets worse results after normalization. For NMF, doing normalization to make each feature with unit variance can get the best results, even adding logarithm before it can get exactly the same results or similar results (since the algorithm gets different results each time). However, doing logarithm after scaling features to unit variance gets worse results. The reason why normalization could help K-means get better results is that different scale of data could affect the distance between data points. Similarly, logarithm transformation is in fact doing the same

thing by decreasing the difference between data points in certain dimension. Therefore, after logarithm transformation, we got better results.

Table 4.1. Clustering scores with different data transformations

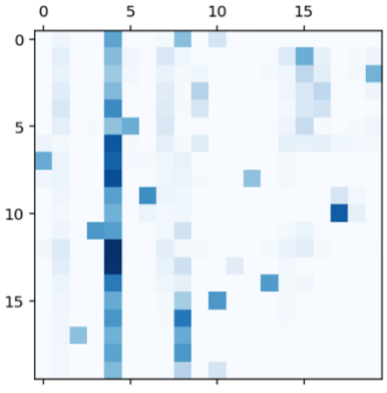
Data	Homogeneity	Completeness	V-measure	Adjusted Rand Index	Adjusted Mutual Info	Contingency Table
SVD + Norm	0.294	0.333	0.312	0.302	0.294	[[1649 2254] [3852 127]]
NMF + Norm	0.450	0.469	0.459	0.504	0.450	[[1024 2879] [3861 118]]
NMF + Log	0.429	0.430	0.429	0.532	0.429	[[3316 587] [478 3501]]
NMF + Log + Norm	0.450	0.469	0.459	0.504	0.450	[[1024 2879] [3861 118]]
NMF + Norm + Log	0.326	0.342	0.334	0.374	0.326	[[251 3652] [2699 1280]]

5. Expand Dataset (problem 5)

To apply the previous steps to a more general scenario, we repeated them on the whole dataset, which includes 20 different labels. This time we got 18846 documents with 31550 terms. First we conducted K-means clustering without transformation and got the following results:

Homogeneity: 0.266
Completeness: 0.355
V-measure: 0.304
Adjusted Rand-Index: 0.052
Adjusted Mutual Information Score: 0.263

Contingency Table:



From the results, we could see that the clustering is not so good. Then we did SVD and NMF transformation and swept the parameter to find the best top *r* principle components for each dimensionality reduction method. To be specific, for SVD feature vectors we swept *r* from 1 to

159, and for NMF feature vectors we swept *r* from 1 to 49, and got the results that for SVD the best *r* is **109** (Figure 5.1) and for NMF the best *r* is **35** (Figure 5.2).

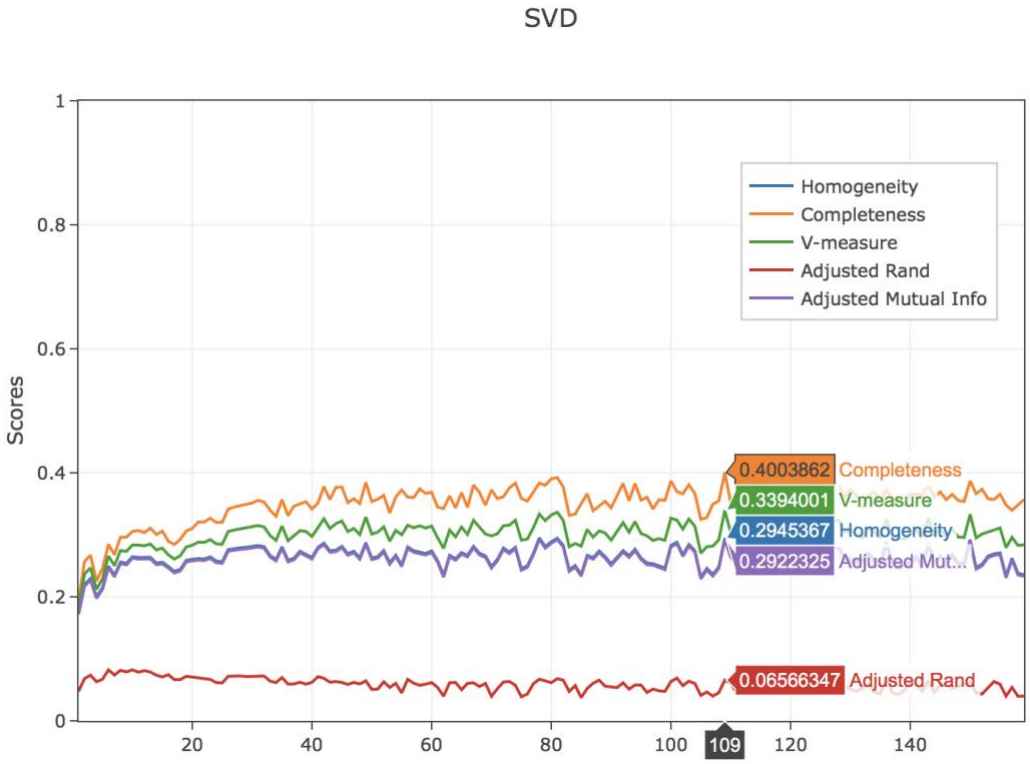


Figure 5.1. Clustering scores of different *r* for SVD (best r = 109)

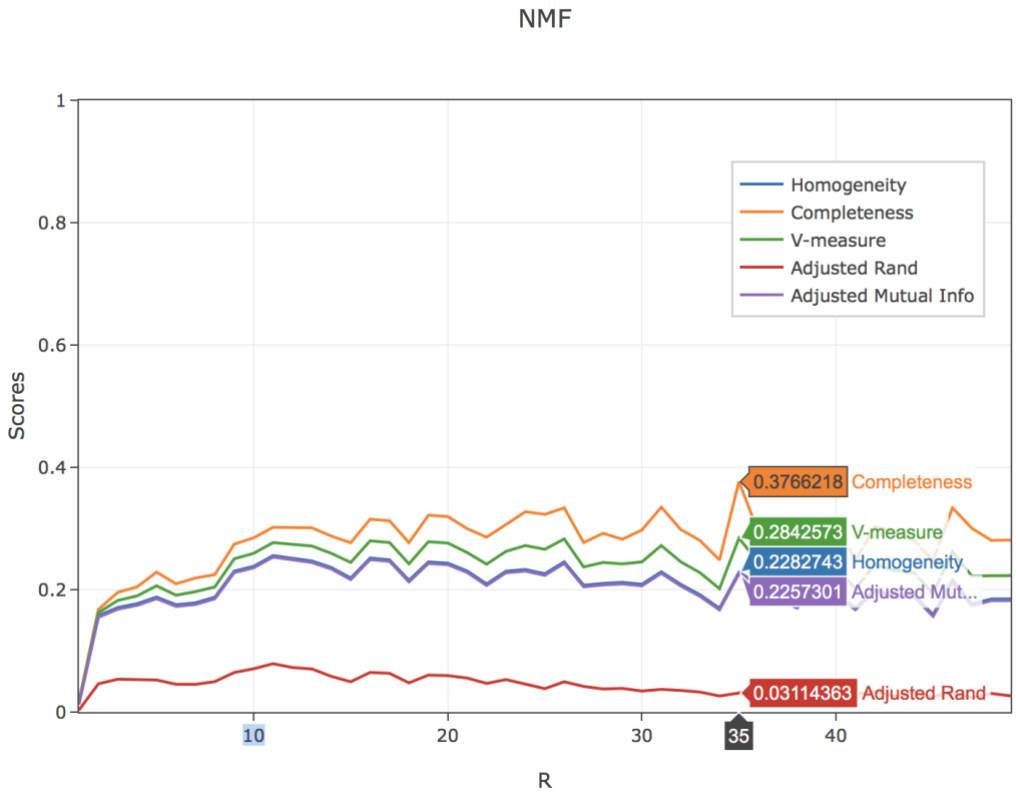
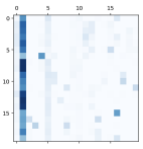
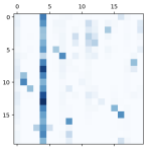
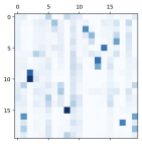
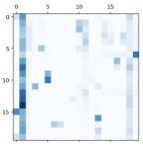
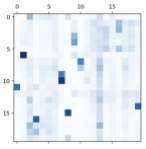


Figure 5.2. Clustering scores of different *r* for NMF (best r = 35)

After that, we tried different combinations of data transformations. (Table 5.1)

Table 5.1. Clustering scores with different data transformations

Data	Homogeneity	Completeness	V-measure	Adjusted Rand Index	Adjusted Mutual Info	Contingency Table
SVD + Norm	0.154	0.275	0.197	0.015	0.151	
NMF + Norm	0.255	0.368	0.301	0.041	0.253	
NMF + Log	0.284	0.290	0.287	0.143	0.281	
NMF + Log + Norm	0.241	0.333	0.280	0.047	0.239	
NMF + Norm + Log	0.254	0.262	0.258	0.121	0.251	

Through the results we could find that the clustering performance is not so good without transformation or normalization. The algorithm tends to cluster a lot of different labels into one class. However, when using logarithm transformation with appropriate parameter, the results get much better. The reason for this is the same as aforementioned that after dimension reduction, the feature vectors become much closer for small values, thus bring negative effect on clustering results, while logarithm transformation to some extent adjust the distribution. As a result, preprocessing feature vectors with logarithm transformation gives out the best clustering results.

6. Additional Work

6.1 Interactive Results

Along with the codes and this report, we also submitted an html file, which is a short interactive report generated from our IPython Notebook. The main features of that html file are:

1. A ‘Show/Hide codes’ button:
   By default, all the codes are hidden when opening this html file, making it clear to read the results for every problems. One can click the button to show all the codes and click another time to hide them again.
2. Interactive plots:
   When hovering over the plots, several functions supported by Plotly will appear on the top right corner, such as downloading, editing, zooming, and panning. Also the exact data on the plot will show when we hovering over the lines. We found the interactive plots are quite helpful for finding the best R according to the clustering scores, especially when the lines are close to each other in one plot. We can either focus on a specific area by selecting it using the mouse or focus on one specific line by clicking on the legend. Finally, just double-click the plot, it will auto-scale back.