ECE 219 Project 2 Report

Clustering

Feb 12th, 2018

Team Members:

Names	UID	email
Evelyn Chen	704332587	chenyiying@ucla.edu
Jack(Chenwei) Gong	005025415	cgpy7@ucla.edu

Introduction:

In this project, we will use clustering algorithm, K means algorithm, to find groups of data that have similar representations in a proper space. We want to find proper representations of the data, perform K-means and evaluate the performance, and moreover, try different preprocess methods and analyze whether they can increase the performance of the clustering.

Problem 1 :

The goal of problem 1 is to build a TF-IDF matrix. We want to find a good representation of the data using min_df = 3 and excluding the stopwords.

By following the procedures from Project1, the documents were transformed into TF-IDF vectors, with min_df = 3 (See details in the image below, line 147).

```
139 if __name__ == "__main__":
140
141
142     categories = ['comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.pc.hardware','comp.sys.mac.hardware','rec.autos','rec.motorcycles','rec.sport.
143     train_data = fetch_20newsgroups(subset='train', categories=categories,shuffle=True, random_state=42)
144     test_data = fetch_20newsgroups(subset='test', categories=categories,shuffle=True, random_state=42)
145     all_data = fetch_20newsgroups(subset='all', categories=categories,shuffle=True, random_state=42)
146     stop_words = text.ENGLISH_STOP_WORDS
147     count_vect = CountVectorizer(stop_words = stop_words, lowercase= True, min_df = 3)
148     tf_transformer = TfidfTransformer(use_idf=False)
149
150     #group the subclasses into 2 superclasses
151     all_target_group = [ int(x / 4) for x in all_data.target]
```

Then, we printed out the dimension of the TF-IDF to be (7882, 27768)
. See details in line 159 as well as its output:

```
152
153     print ('question 1: ==============')
154
155     vectorizer = TfidfVectorizer(
156                                  min_df=3, stop_words=stop_words,
157                                  )
158     tfidf_transformer = vectorizer.fit_transform(all_data.data)
159     print ("dimension of tfidf: ", tfidf_transformer.shape)
```

```
JacktekiMacBook-Pro:project2 JackKalman$ python3 evelyn.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/JackKalman/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
question 1: ==============
dimension of tfidf:  (7882, 27768)
```

Problem 2 (a):

The goal of problem 2 is to apply K-means clustering with k = 2 with the TF-IDF data. First, we look at the contingency matrix to see the clustering result.

Since there were 8 categories provide by the dataset, and the clustering factor k = 2. We figured that this should be a 2 by 2 contingency matrix, where dimension 2 stands for two different classes (‘Computer Technology’ and ’Recreation’). To accomplish this, we categorize the targets with label 0-3(‘Computer Technology’ sub classes) into 0, and targets with label 4-7(’Recreation’ sub classes) into 1, using integer division (line 151).

```
142     categories = ['comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.pc.hardware','comp.sys.mac.hard
143     train_data = fetch_20newsgroups(subset='train', categories=categories,shuffle=True, random_state=42)
144     test_data = fetch_20newsgroups(subset='test', categories=categories,shuffle=True, random_state=42)
145     all_data = fetch_20newsgroups(subset='all', categories=categories,shuffle=True, random_state=42)
146     stop_words = text.ENGLISH_STOP_WORDS
147     count_vect = CountVectorizer(stop_words = stop_words, lowercase= True, min_df = 3)
148     tf_transformer = TfidfTransformer(use_idf=False)
149
150     #group the subclasses into 2 superclasses
151     all_target_group = [ int(x / 4) for x in all_data.target]
```

Then, we imported KMeans from sklearn.cluster and listed out the labels for each point, and put it into the confusion_matrix calculator, provided by sklearn.metrics (see details in the image below).

```
161     print ('question 2: ==============')
162
163     km = KMeans(n_clusters=2, max_iter = 1000, random_state=42).fit(tfidf_transformer)
164
165     print ('contingency matrix:')
166     print (confusion_matrix(all_target_group, km.labels_))
167     print_five_measures(all_target_group, km.labels_)
168
```

Finally, the output appeared to be:

```
question 2: ==============
contingency matrix:
[[   4 3899]
 [1717 2262]]
```

Problem 2 (b)

In part b, we want to try different measures to evaluate the clustering results. We will examine measures including the homogeneity score, the completeness score, the V-measure, the adjusted rand score, and the adjusted mutual info score. Homogeneity is a measure of how pure the cluster is; completeness is whether all data points of a class are assigned to the same cluster. V-measure is the harmonic average of homogeneity and completeness score. The adjusted rand index, similar to accuracy, is the similarity between clustering labels and true labels. The adjusted mutual information score is the mutual information between cluster label and ground truth label distribution.

Following the procedure from 2(b), we calculated the homogeneity score, the completeness score, the V-measure, the adjusted Rand score and the adjusted mutual info score respectively, using the function provided by sklearn.metrics;

```
122  def print_five_measures (target, predicted):
123      print ('homogeneity score:')
124      print (homogeneity_score(target, predicted))
125
126      print ('completeness score:')
127      print (completeness_score(target, predicted))
128
129      print ('V-measure:')
130      print (v_measure_score(target, predicted))
131
132      print ('adjusted rand score:')
133      print (adjusted_rand_score(target, predicted))
134
135      print ('adjuted mutual info score:')
136      print (adjusted_mutual_info_score(target, predicted))
137
```

The output appeared as following:

```
homogeneity score:
0.25341287993596523
completeness score:
0.3346772445110614
V-measure:
0.2884303641736151
adjusted rand score:
0.18054602343005732
adjuted mutual info score:
0.25334452160645765
```

Problem 3(a)(i):

In part 3, we want to find a better representation on how K-means clustering works by preprocessing our data. We use Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF). First, we inspect top singular values and see the ratio of variance of original data is retained after dimensionality reduction. Then, we report the plot of percent of variance of the top r principal components.

The ratio of variance is calculated by

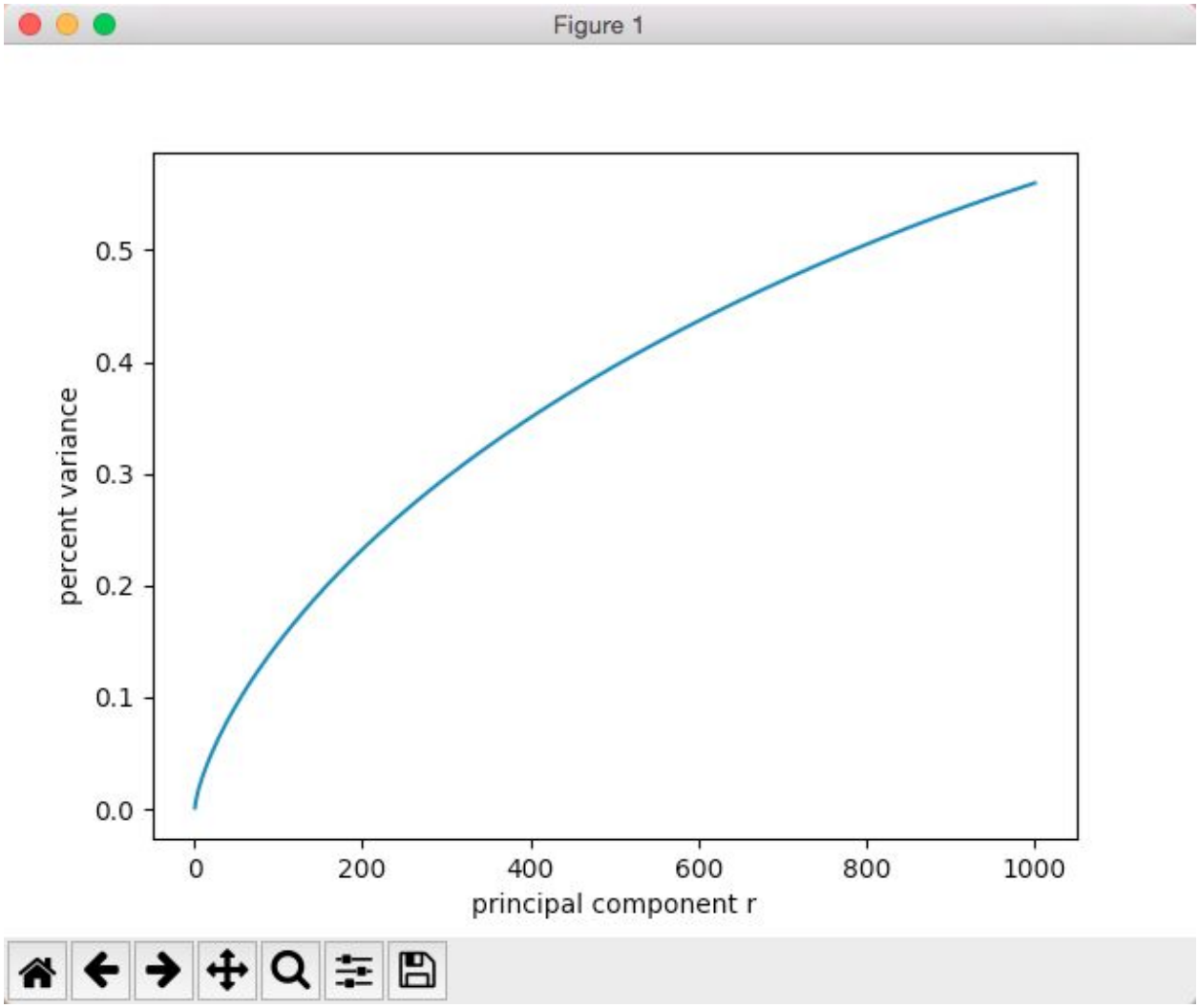
$$ratio = \frac{varience\_retained}{total\_varience} = \frac{\sum_{i=1}^{r} S_{ii}^2}{\sum_{i=1}^{r_{max}} S_{ii}^2}$$

In this project, we utilize the function svd.explained_variance_ratio_ to find the ratio of variance, which is slightly different from the equation above, but it is also sufficient to use here.

The range of r is from r = 1 to 1000.

```
45  def plot_percent_variance(tfidf_transformer):
46      svd = TruncatedSVD(n_components=1000) #change to 1000
47      svd.fit_transform(tfidf_transformer)
48      ratio = svd.explained_variance_ratio_
49      singular_values = svd.singular_values_
50
51      #retained value
52      sum_arr = []
53      for k in range(1,1001): #change to 1001
54          sum = 0.0
55          for i in range(k):
56              sum = sum + ratio[i]
57          sum_arr.append(sum)
58
59      x_values = range(1,1001)
60
61      plt.plot(x_values, sum_arr)
62      plt.xlabel('principal component r')
63      plt.ylabel('percent variance')
64      plt.show()
```

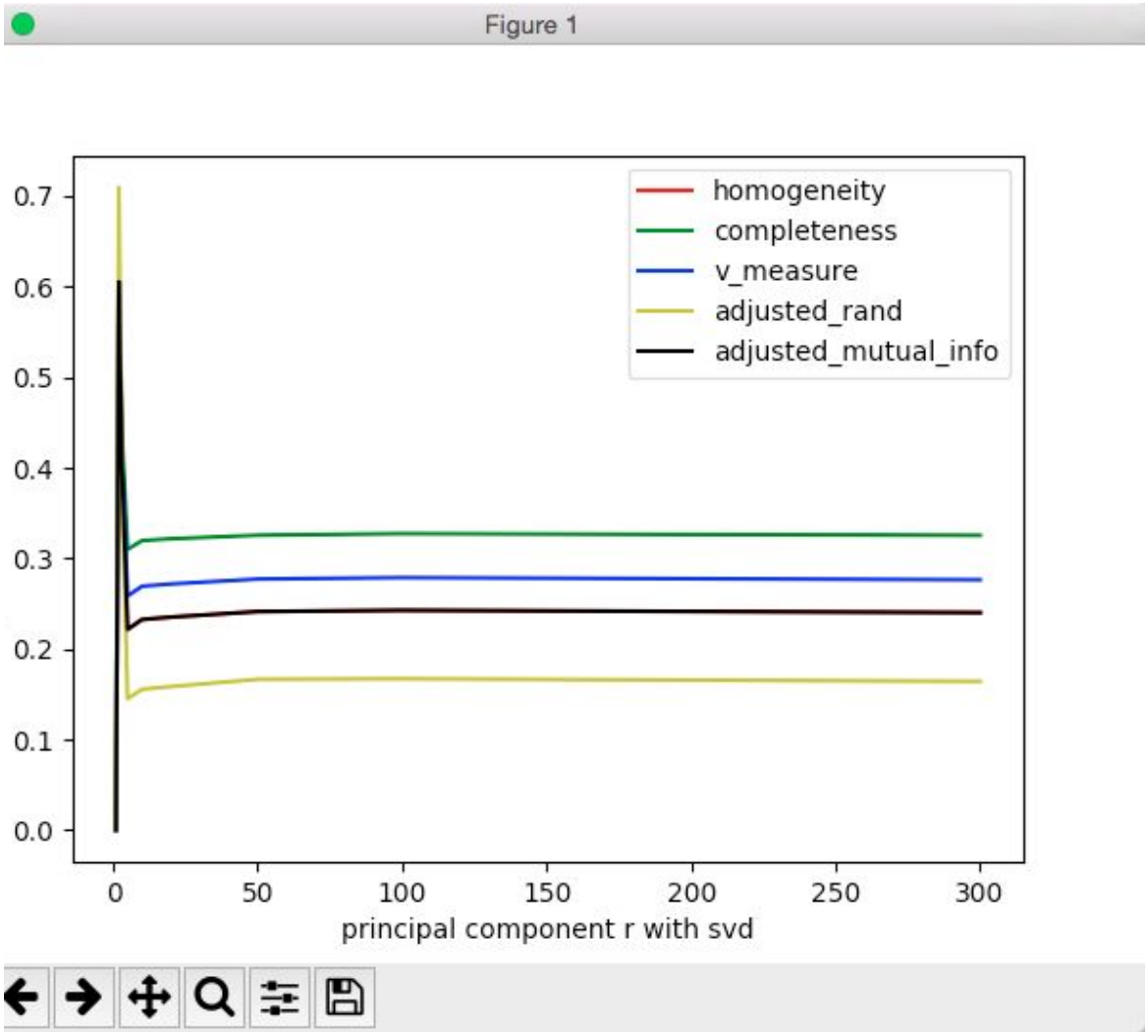
The output is shown below:



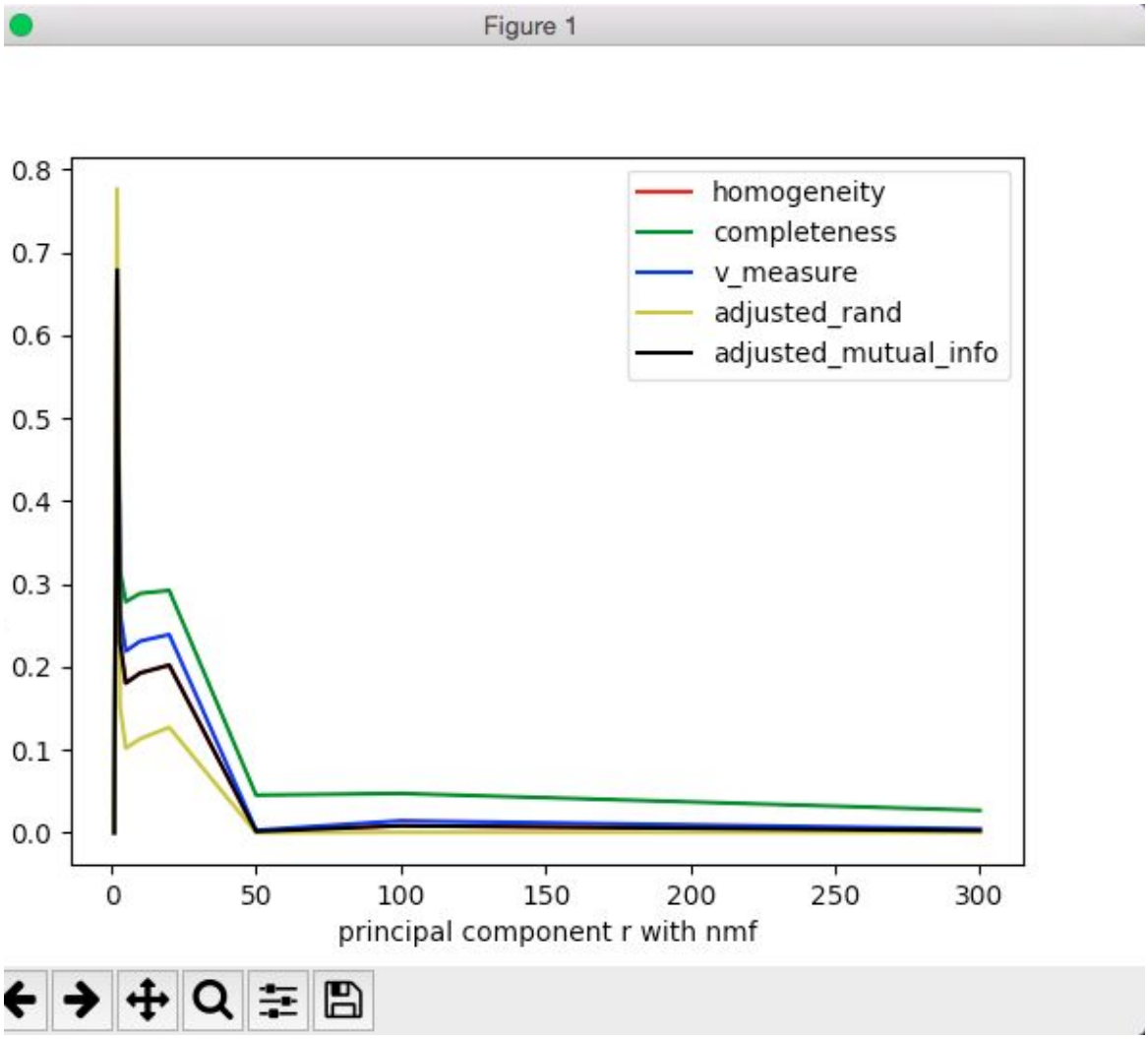
Based on the observation, the gradient is gradually decreasing. This is because the Singular value in the matrix is arranged in a descending order, from the most significant (core) information to the least significant ones. As r increases(increasing the dimension therefore more information is added), more information will be included, but not as crucial as before, because the Singular Value added is not as significant. Therefore the gradient of the curve is gradually decreasing and will slowly, eventually approximate to 1.

Problem 3(a)(ii)

Two different methods were used to reduce the dimension of the data, truncated SVD and NMF. The ratio r was set to be [1, 2, 3, 5, 10, 20, 50, 100, 300], and the homogeneity score, the completeness score, the V-measure, the adjusted Rand score and the adjusted mutual info score were plotted with respect to each r given. The output for Truncated SVD method appeared as following:



Whereas the output of NMF, appeared to be:



As for the comparison between the contingency matrix, a table was made to better visualize the contrast between these two methods, as shown below:

r	SVD	NMF
1	[[2200 1703] [2323 1656]]	[[2200 1703] [2323 1656]]
2	[[3671 232] [380 3599]]	[[3594 309] [158 3821]]
3	[[3868 35] [1428 2551]]	[[3899 4] [2396 1583]]
5	[[3898 5] [2435 1544]]	[[3898 5] [2677 1302]]
10	[[3 3900] [1597 2382]]	[[4 3899] [1353 2626]]
20	[[3900 3] [2354 1625]]	[[21 3882] [1365 2614]]
50	[[3900 3] [2354 1625]]	[[3893 10] [3979 0]]
100	[[3 3900] [1653 2326]]	[[0 3903] [25 3954]]
300	[[3900 3] [2340 1639]]	[[3146 757] [3006 973]]

From the chart above and from the two graphs on different r, we can see the SVD performs the best with r = 2 and NMF performs the best with r = 3. The two charts that plot measures against r for SVD and NMF peak at r=2 and r=3 relatively.

Q: How do you explain the non-monotonic behavior of the measures as r increases?
A: Starting from the origin, where the dimensions were initially very small, very little information is provided to the set and thus, a slight increment on the dimension will largely improve the clustering result. However, if we continue to increase the dimensions, we're providing more and more information, in details, and ask the program to fit all the data based on those details. This can be extremely difficult if the dimensionality is extremely high, since there are too much information provided as a mean to fit into different clusters. From a more professional point of view, referencing the paper in (https://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a-good-metric-in-high-dimensions), the r is considered a Gaussian HyperSphere, where its volumn is proportional to the r value. If the Gaussian HyperSphere is too large, little will the program know which clusters the information belongs to.

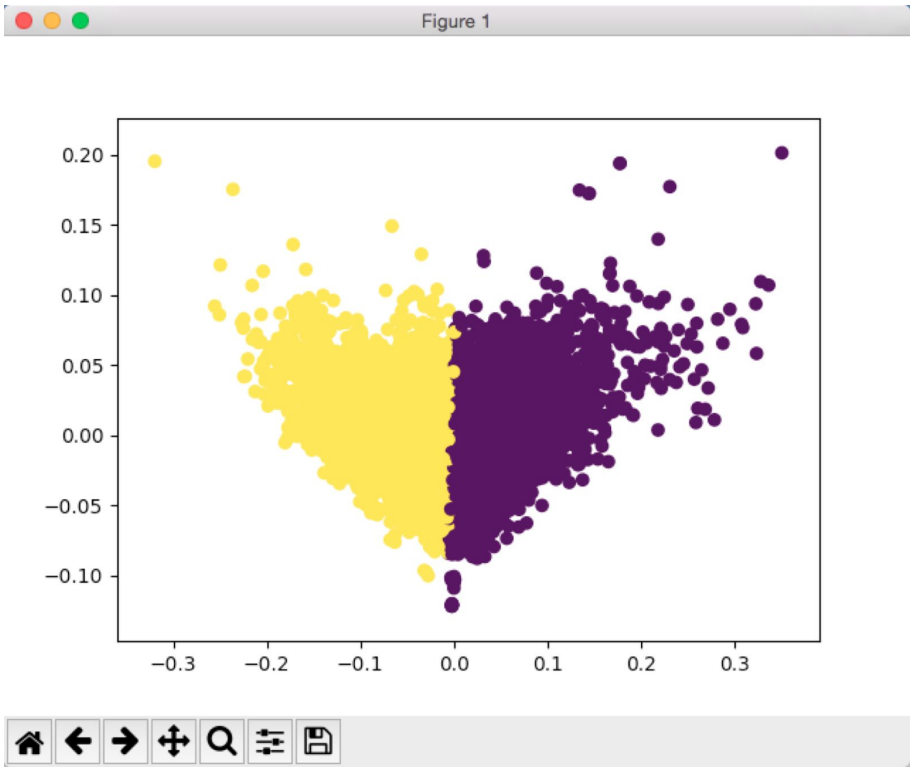
Problem 4

a)

Here, we visualize the performance of the best clustering results. We project data vectors onto 2 d plane with color-coding classes.

From the previous steps, we found that SVD performs at its finest when r = 2. Then the final data were selected to be projected on a 2 dimensional plane, where the 2 dimensions are the most significant attributes determined by PCA. Codes and figure are shown below:

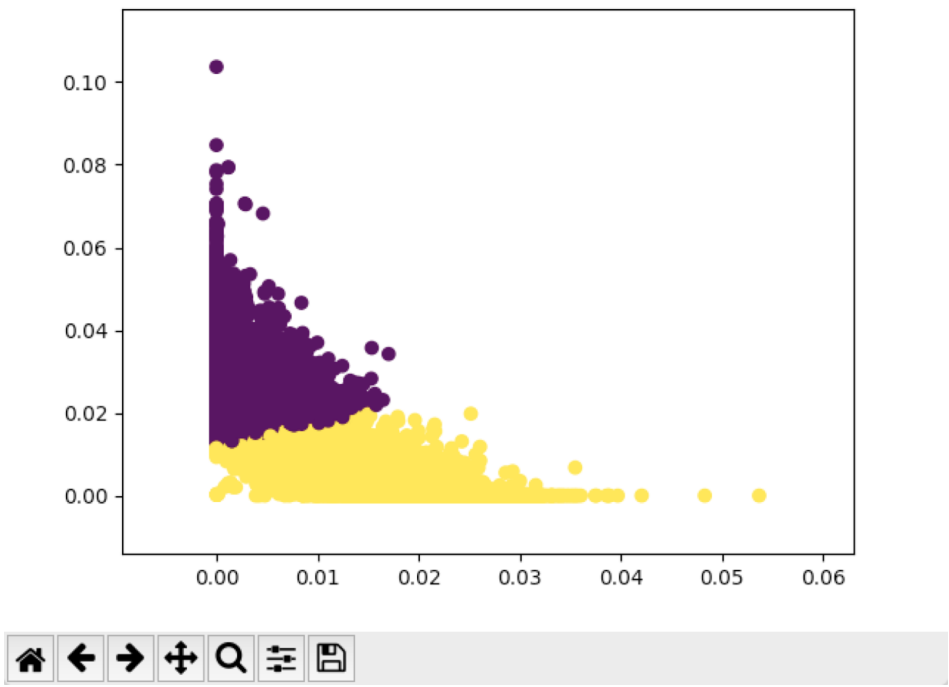
```
def visualizePerformance(tfidf_transformer, target, k, n): #target, num_cluster, n_component
    print("svd at its best score r =",n)
    svd = TruncatedSVD(n_components=n) #best score is 2
    svd_data = svd.fit_transform(tfidf_transformer)
    km = KMeans(n_clusters=k, n_init = 30, random_state=42).fit_predict(svd_data)
    pca = PCA(n_components=n).fit_transform(svd_data)
    plt.scatter(pca[:,0], pca[:,1], c = km)
    plt.show()
```



NMF, on the other hand, has its best performance on r = 3, as marked out from the previous step. The clustering result, after applying NMF, followed by its implementation is showing below:

```
print("nmf at its best score r = ",n)
nmf = NMF(n_components = n, init = 'random', random_state = 42) #best score is
nmf_data = nmf.fit_transform(tfidf_transformer)
km = KMeans(n_clusters=k, n_init = 30, random_state=42).fit_predict(nmf_data)
plt.scatter(nmf_data[:,0], nmf_data[:,1], c = km)
plt.show()
```

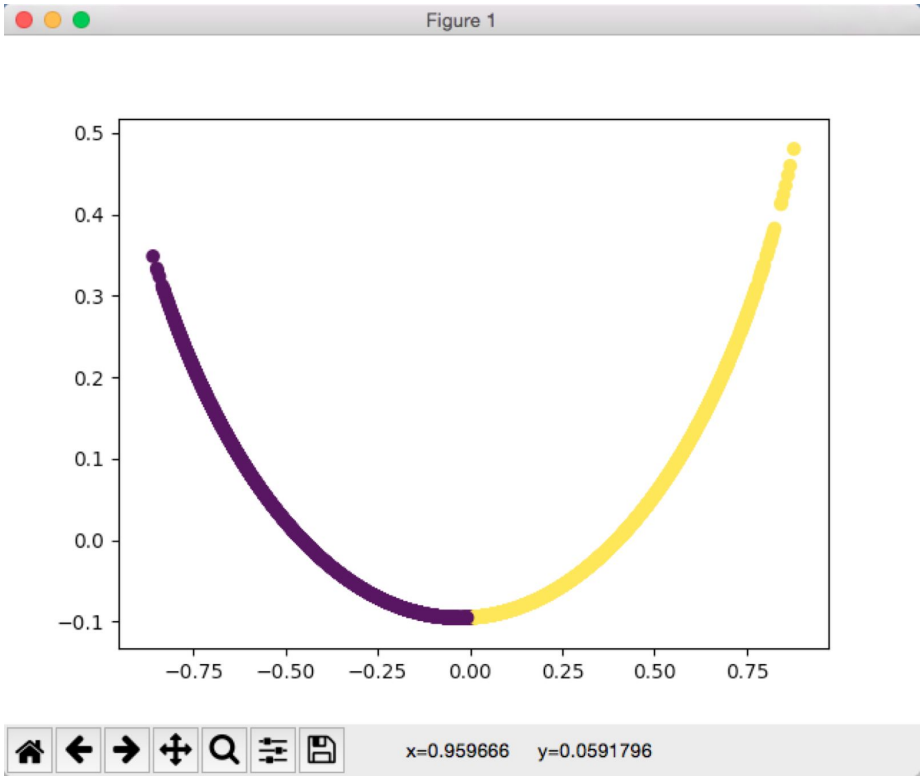
Figure 1



b)
(1)

We normalized svd before performing the k means clustering, and the source code, image, and contingency matrix featured with all five scores are showing as the following:

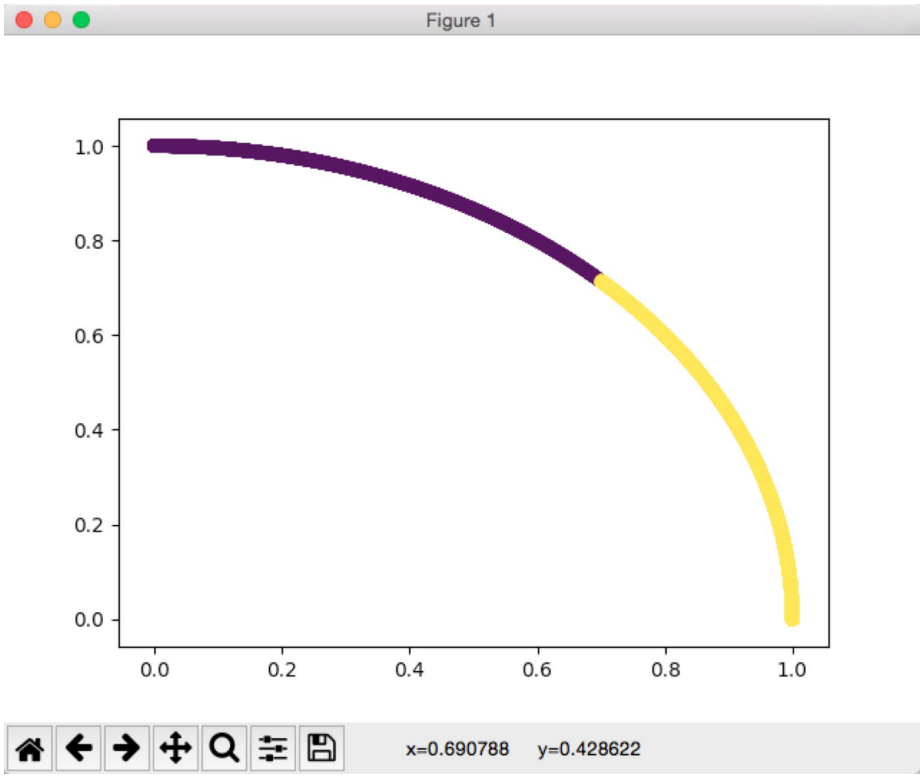
```
167     print ('part b --------')
168     # svd normalzed
169     print('svd normalized')
170
171     svd_norm = normalize(svd_data)
172     kmeans = KMeans(n_clusters=2, n_init=30)
173     km = kmeans.fit_predict(svd_norm)
174     pca = PCA(n_components=2).fit_transform(svd_norm)
175     plt.scatter(pca[:,0], pca[:,1], c = km)
176     plt.show()
177     print('contingency matrix: ')
178     print(confusion_matrix(all_target_group, kmeans.labels_))
179     print_five_measures(all_target_group, kmeans.labels_)
```



```
part b --------
svd normalized
contingency matrix:
[[ 291 3612]
 [3666  313]]
homogeneity score:
0.609878880023441
completeness score:
0.6098452284949754
V-measure:
0.6098620537949945
adjusted rand score:
0.7169317077667118
adjuted mutual info score:
0.6098095116879794
```

Similarly, NMF was performed after normalization, with the factor of r = 2 as its finest.

```
181     print('nmf normalzed')
182
183     nmf_norm = normalize(nmf_data)
184     kmeans = KMeans(n_clusters=2, n_init=30)
185     km = kmeans.fit_predict(nmf_norm)
186     plt.scatter(nmf_norm[:,0], nmf_norm[:,1], c = km)
187     plt.show()
188     print('contingency matrix: ')
189     print(confusion_matrix(all_target_group, kmeans.labels_))
190     print_five_measures(all_target_group, kmeans.labels_)
191
```




```
nmf normalzed
contingency matrix:
[[3763  140]
 [ 265 3714]]
homogeneity score:
0.7110049364423556
completeness score:
0.7112072873246874
V-measure:
0.7111060974883845
adjusted rand score:
0.8050044807923289
adjuted mutual info score:
0.7109784788605964
```

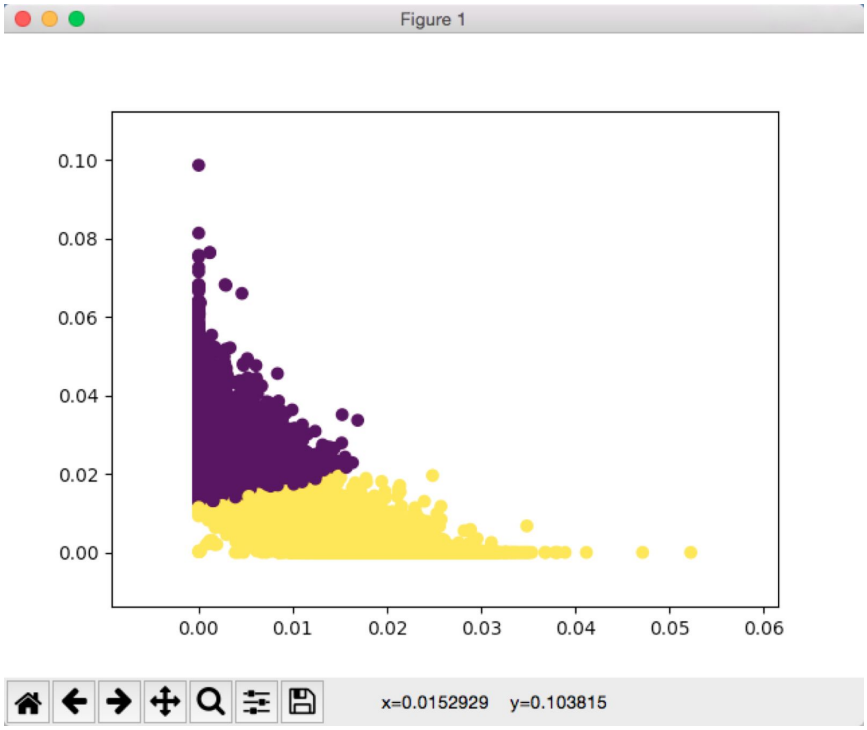
From the five measures from SVD and NMF after normalization, we see that the measures are similar to pure SVD and NMF as the adjusted rand score were around 0.7 and 0.8 before normalization as well.

(2)

The glorious logarithmic transformation was applied on top of the NMF data, uncovering the clustering image, contingency matrix as well as the five measuring scores:

The logarithmic operation can bring the extreme high values down to normal value (so the data sets won't contain as much scattered points), thus fitting more values into the cluster. Therefore, the clustering performance is increased by the log transformation.

```
194     print('applying log transformation after NMF hear:')
195
196     logTransform = FunctionTransformer(np.log1p) #(log10, log2, log1p, emath.log
197     nmf_log = logTransform.transform(nmf_data)
198     kmeans = KMeans(n_clusters=2, n_init=30)
199     km = kmeans.fit_predict(nmf_log)
200     plt.scatter(nmf_log[:,0], nmf_log[:,1], c = km)
201     plt.show()
202     print('contingency matrix: ')
203     print(confusion_matrix(all_target_group, kmeans.labels_))
204     print_five_measures(all_target_group, kmeans.labels_)
```

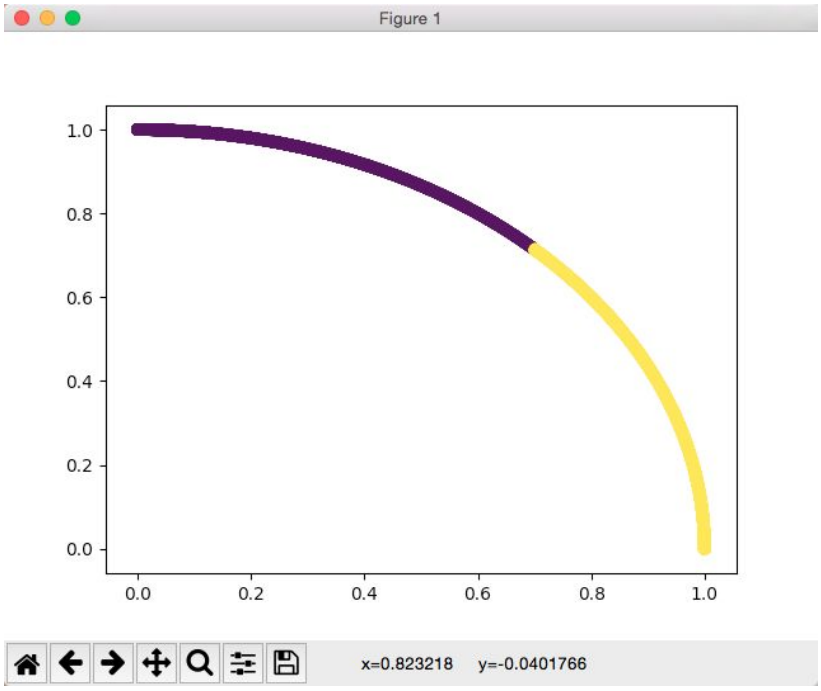


```
applying log transformation after NMF hear:
contingency matrix:
[[3141  762]
 [  36 3943]]
homogeneity score:
0.5824804912485355
completeness score:
0.5987770903994707
V-measure:
0.5905163770930725
adjusted rand score:
0.6359817767144438
adjuted mutual info score:
0.5824422668422519
```

(3)

As we fumbling forward, the logarithmic transformation was again, applied on the NMF, except for this time, the result generated after the transformation was normalized.

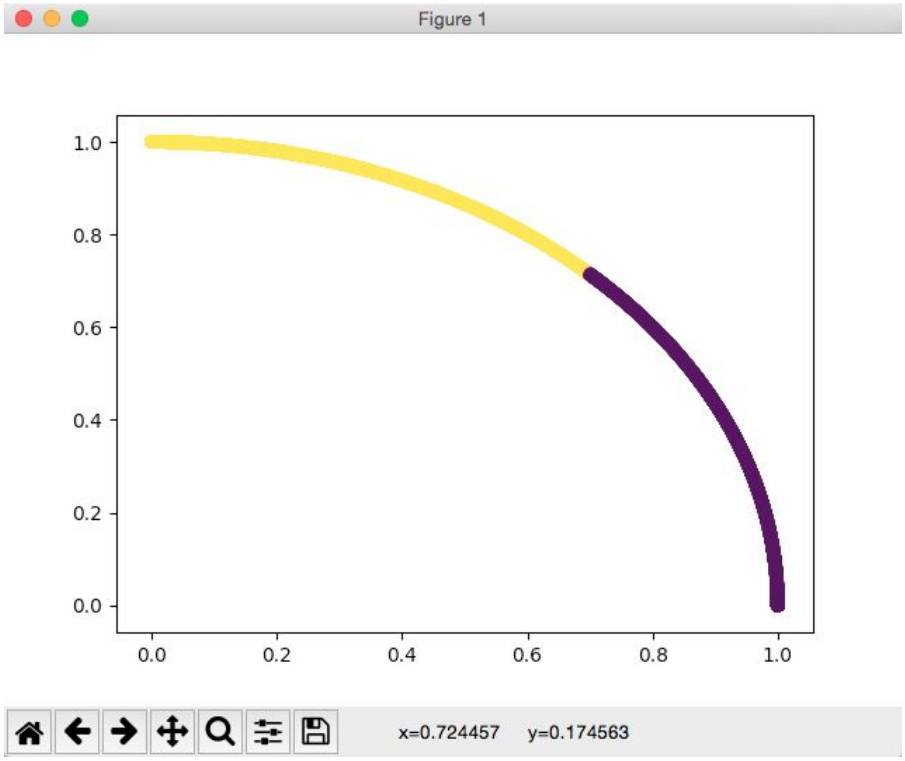
```
207   # 3rd bullet
208       print('log then norm')
209
210       nmf_log = logTransform.transform(nmf_data)
211       nmf_log_norm = normalize(nmf_log)
212       kmeans = KMeans(n_clusters=2, n_init=30)
213       km = kmeans.fit_predict(nmf_norm)
214       plt.scatter(nmf_norm[:,0], nmf_norm[:,1], c = km)
215       plt.show()
216       print('contingency matrix: ')
217       print(confusion_matrix(all_target_group, kmeans.labels_))
218       print_five_measures(all_target_group, kmeans.labels_)
```




```
log then norm
contingency matrix:
[[ 140 3763]
 [3714  265]]
homogeneity score:
0.7110049364423556
completeness score:
0.7112072873246874
V-measure:
0.7111060974883845
adjusted rand score:
0.8050044807923289
adjuted mutual info score:
0.7109784788605964
```

Finally, the NMF data was first normalized, then the logrithmic transformation was applied.

```
220     print('norm then log')
221
222     nmf_norm = normalize(nmf_data)
223     nmf_norm_log = logTransform.transform(nmf_norm)
224     kmeans = KMeans(n_clusters=2, n_init=30)
225     km = kmeans.fit_predict(nmf_norm)
226     plt.scatter(nmf_norm[:,0], nmf_norm[:,1], c = km)
227     plt.show()
228     print('contingency matrix: ')
229     print(confusion_matrix(all_target_group, kmeans.labels_))
230     print_five_measures(all_target_group, kmeans.labels_)
```




```
norm then log
contingency matrix:
[[3763  140]
 [ 265 3714]]
homogeneity score:
0.7110049364423556
completeness score:
0.7112072873246874
V-measure:
0.7111060974883845
adjusted rand score:
0.8050044807923289
adjuted mutual info score:
0.7109784788605964
```

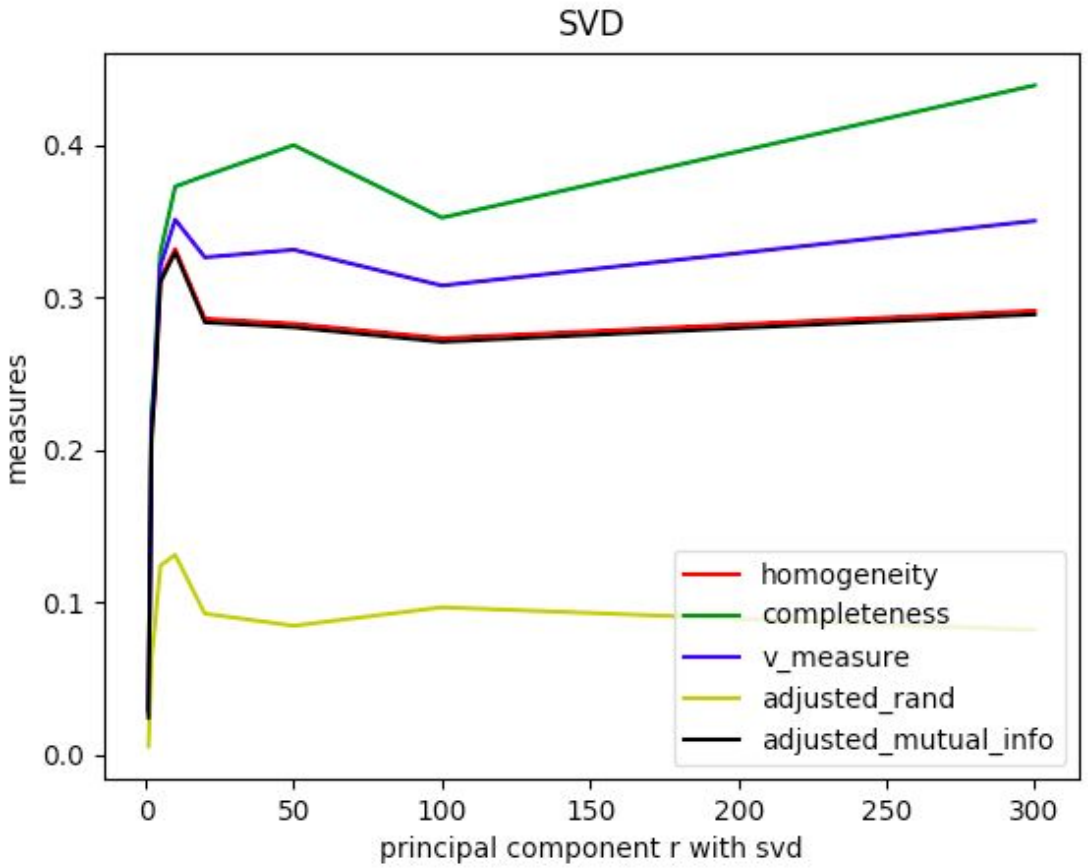
The clusterings for norm then log and log then norm are interesting as shown from the graph. THe measures seem similar to original clustering with around 0.8 for adjusted rand score..etc.

Problem 5

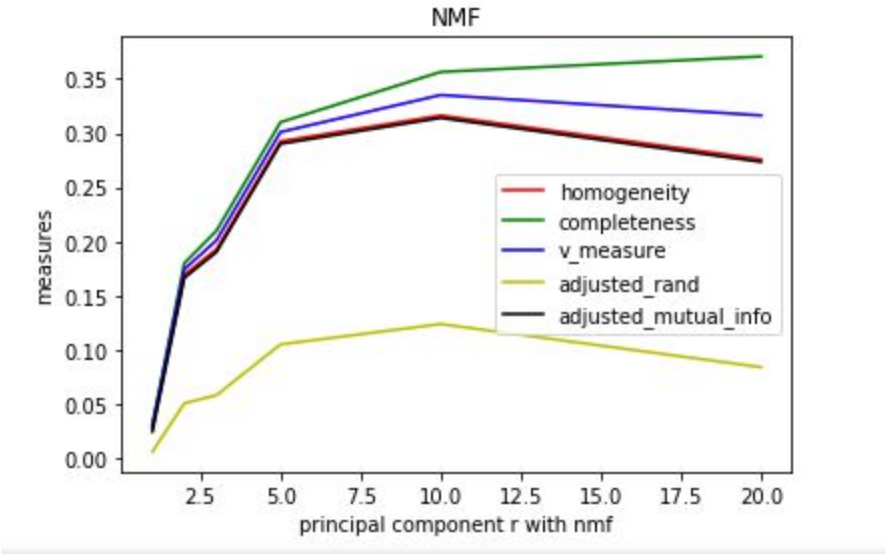
Previously, we tried to fit 8 categories into two clusters. In problem 5, we want to expand the dataset from 8 categories to 20 categories. We want to fit 20 categories into 20 clusters. We will include all the documents and terms with also min_df = 3 and stop_words used in previous sections.

1) 20 clusters for SVD and NMF for different dimensionalities

Below is the five measures for SVD for 20 clusters. As the diagram below shows, the measures mostly peak at r = 10, with only completeness peaks around r = 50. We would say r = 10 is the best dimensionality for SVD for 20 clusters.



Next, we want to find the five measures for NMF for 20 clusters.



We use r=1,2,3,5,10,20 for NMF

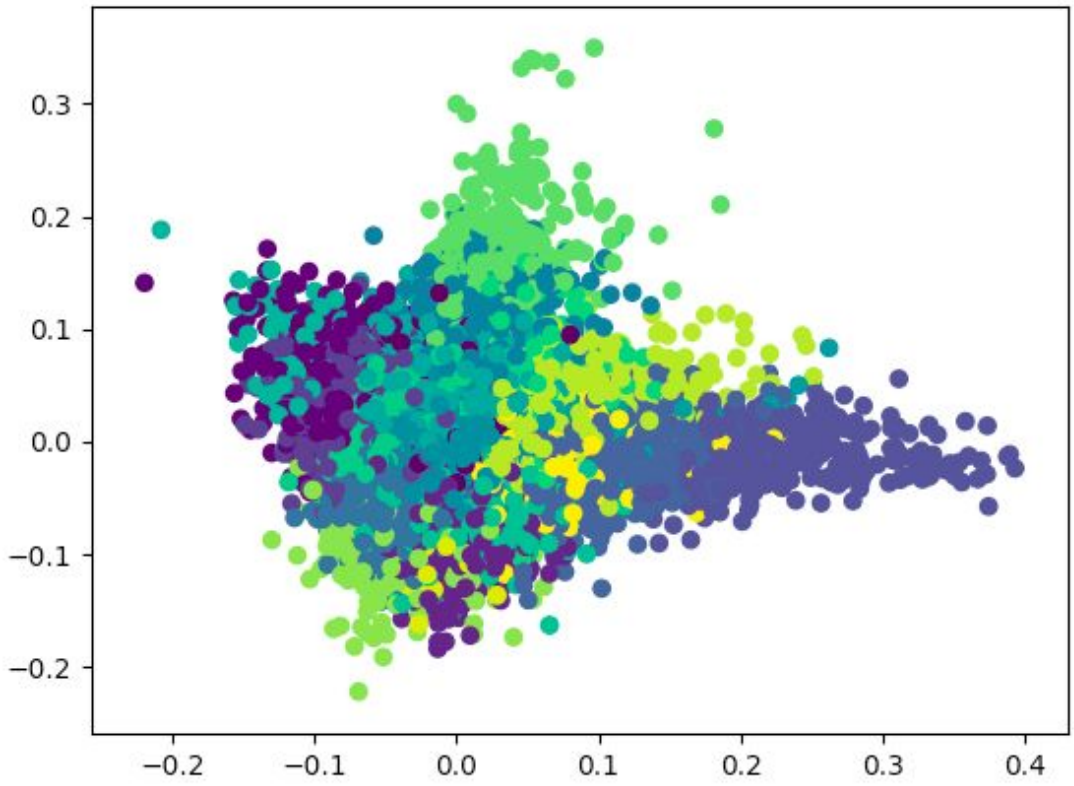
This is because in our personal computer, there is “segfaults” due to lack of memory and internal error Mac has. However, we can see that in general, the measures peak at around r = 10 for NMF for 20 clusters.

2) Visualize data for 20 clusters for SVD and NMF with different transformations outlined in problem 4

We will output diagrams for many scenarios: pure SVD, pure NMF, SVD normalized, NMF normalized, apply log transformation after NMF, log then norm NMF, and norm then log NMF. We use the methods explained in problem 4, just to change it to 20 clusters here.

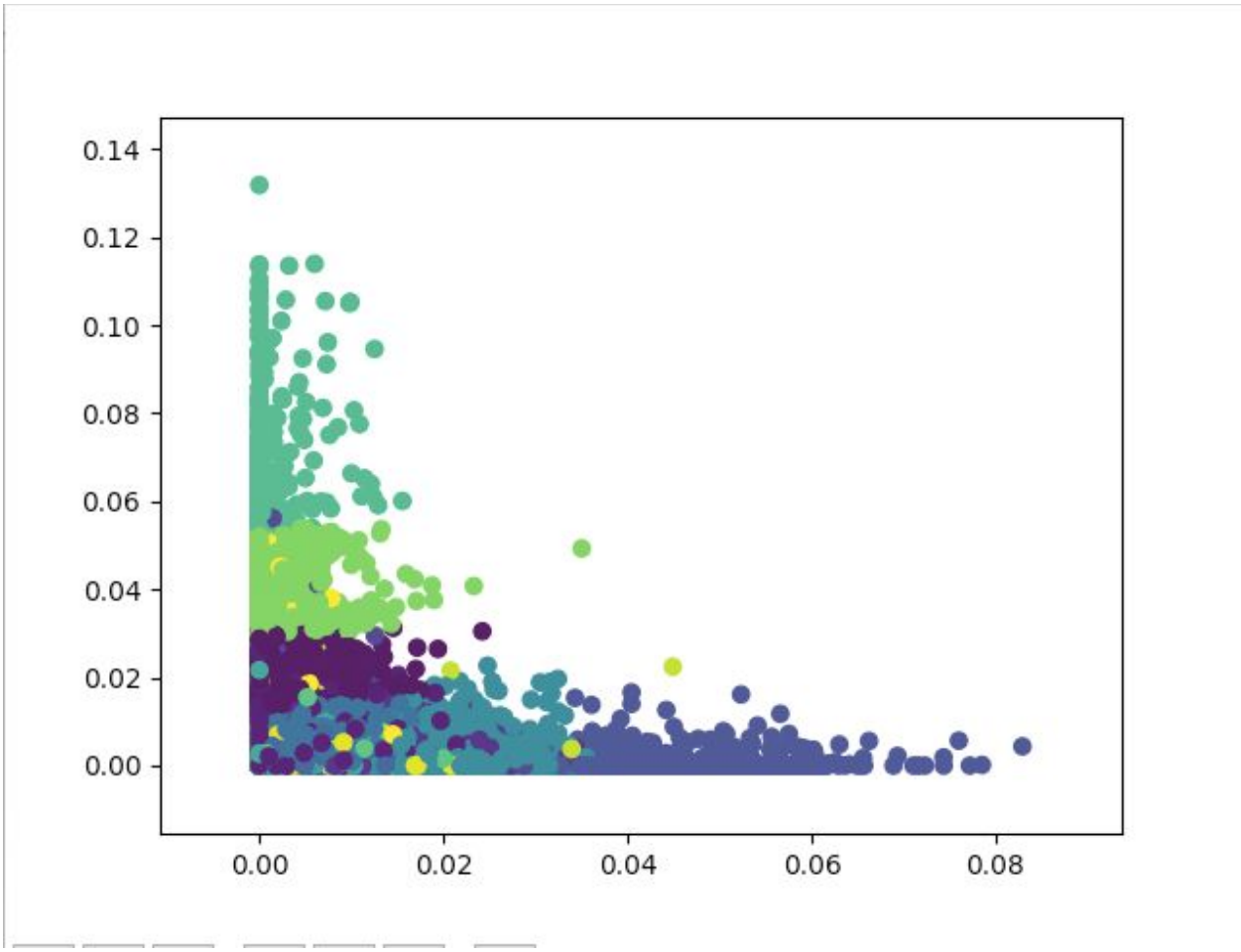
<u>Pure SVD</u>

Below is the graph for pure SVD. It clusters well as same colors are in the same area.



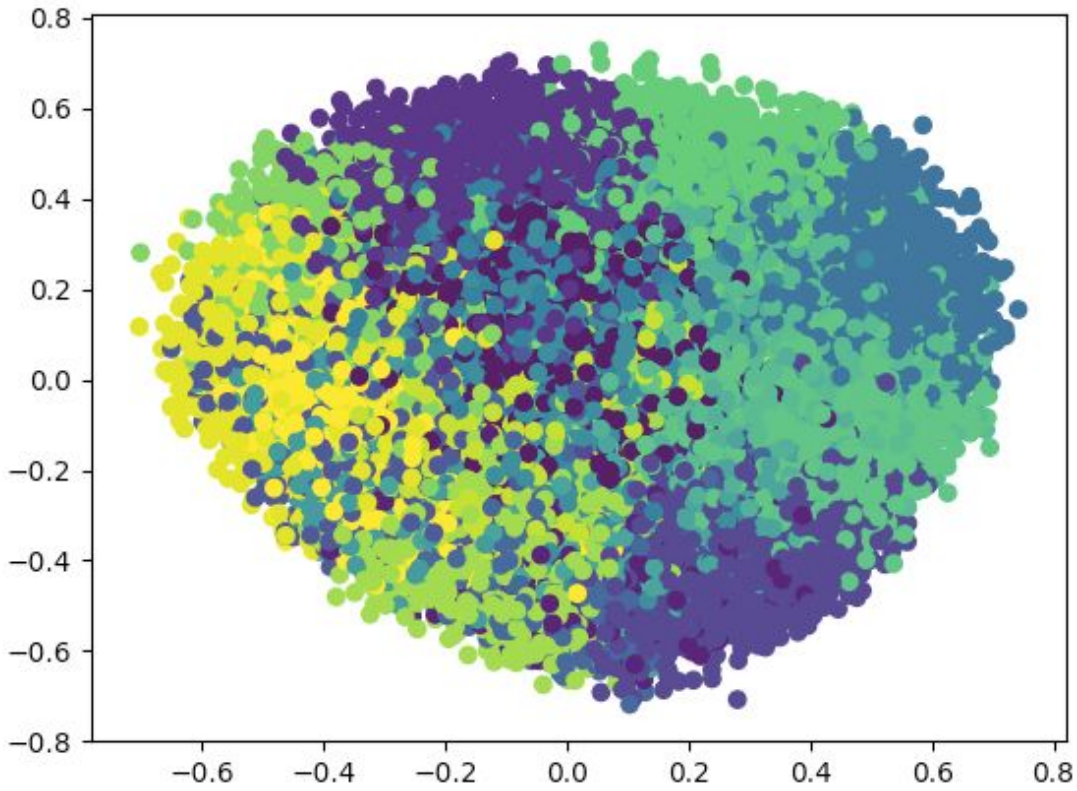
Pure NMF

Below is pure NMF. It also clusters well as same colors are in the same area.



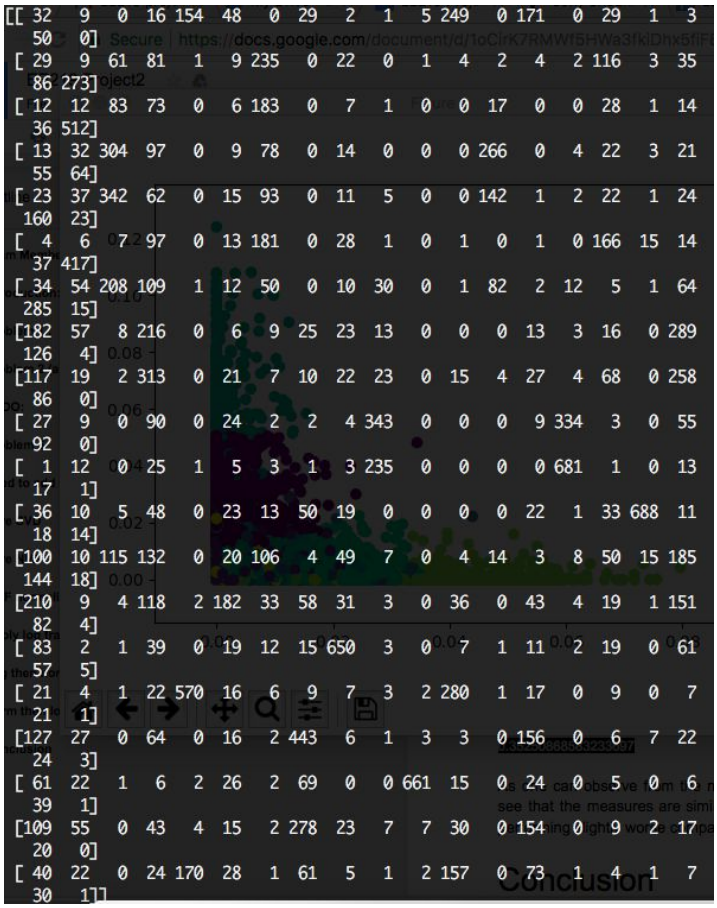
SVD normalized

When SVD is normalized, it looks more in a round-shape than star-shaped (pure SVD). It seems like pure SVD is doing a better job in clustering.



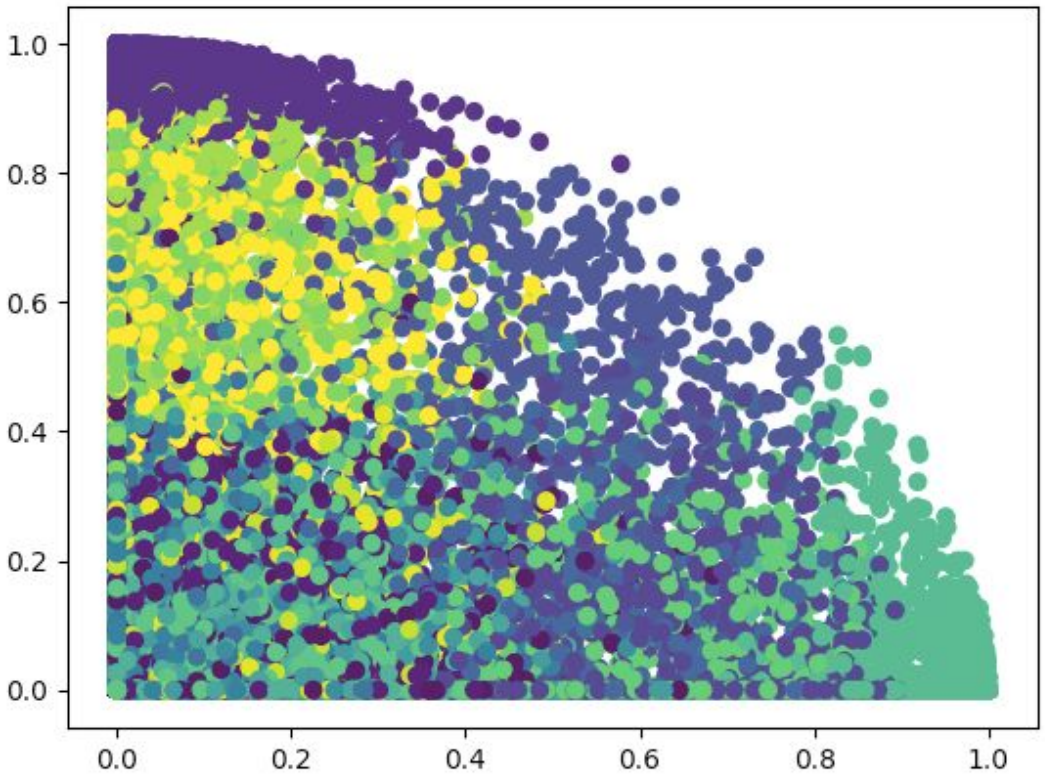
Measures:
homogeneity score:
0.39761880850226633
completeness score:
0.4045372525014534
V-measure:
0.4010481953677153
adjusted rand score:
0.2282443783016558
adjuted mutual info score:
0.39567425095233305

Contingency matrix:



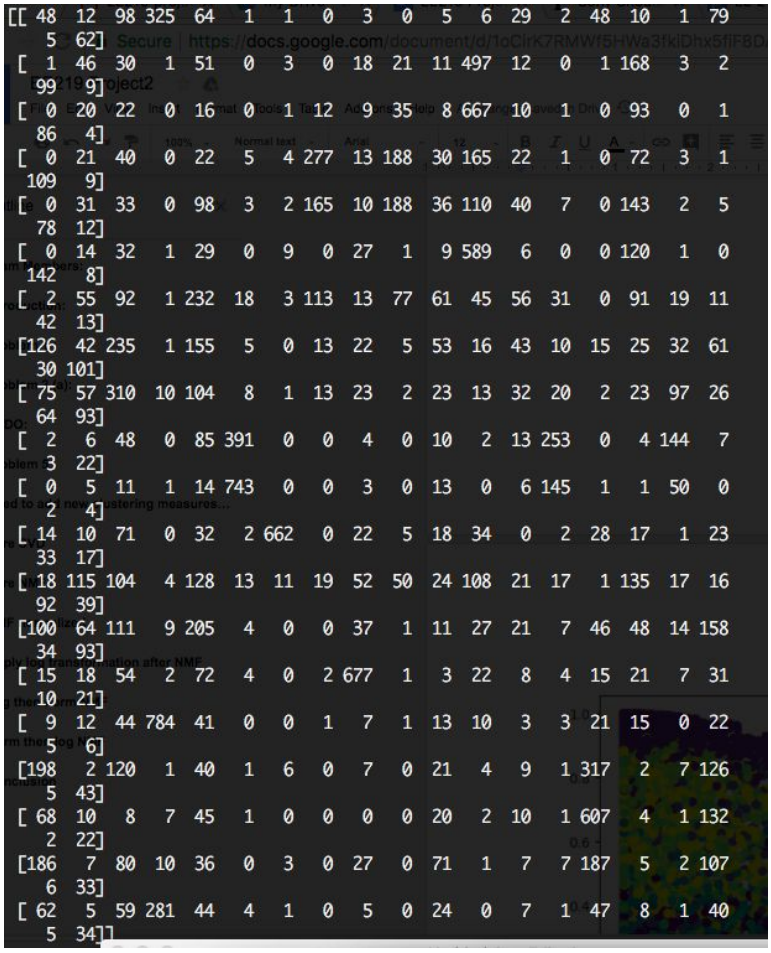
NMF normalized

Below is graph for NMF normalized.



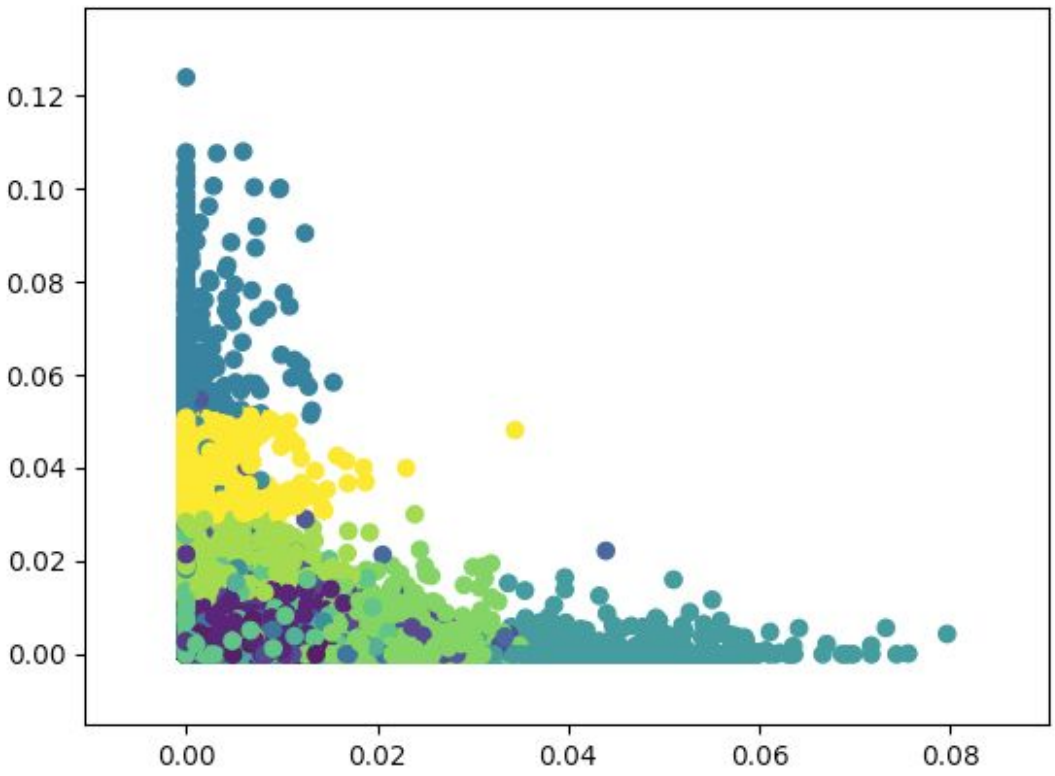
homogeneity score:
0.3561209494265138
completeness score:
0.36902627025343726
V-measure:
0.36245877294815143
adjusted rand score:
0.2091662432474998
adjusted mutual info score:
0.35404170973769694

Contingency matrix:



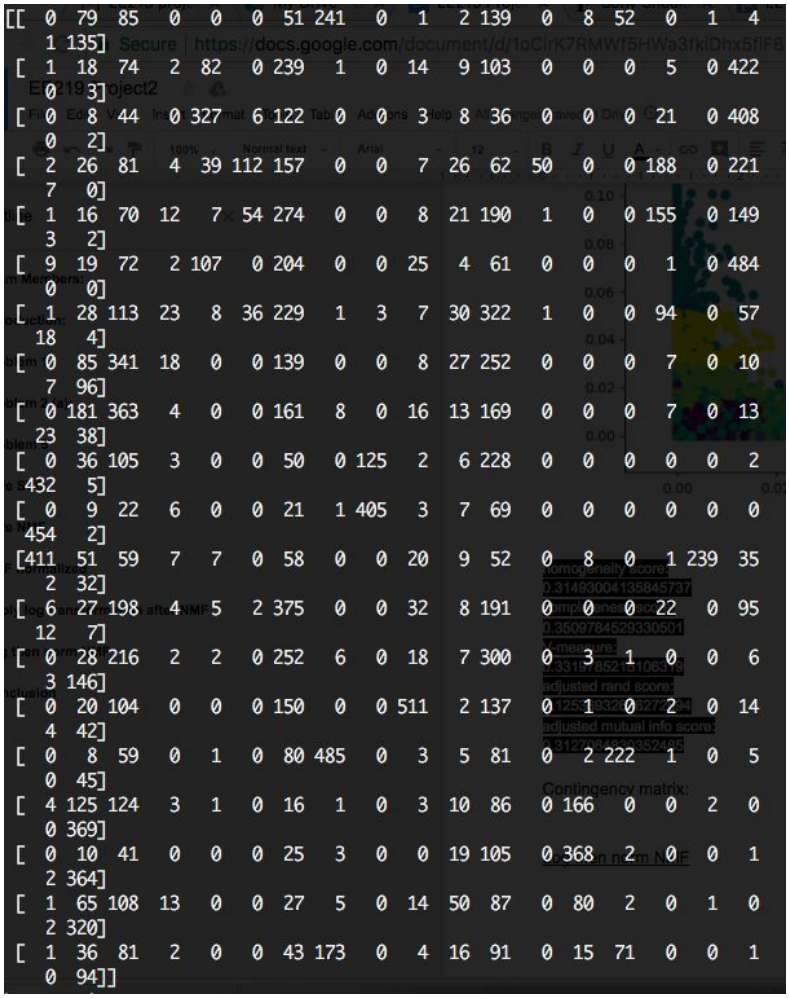
```
[[ 48  12  98 325  64   1   1   0   3   0   5   6  29   2  48  10   1  79
   5  62]
 [  1  46  30   1  51   0   3   0  18  21  11 497  12   0   1 168   3   2
  99  19]
 [  0  20  22   0  16   0   1  12   9  35   8 667  10   1   0  93   0   1
  86   4]
 [  0  21  40   0  22   5   4 277  13 188  30 165  22   1   0  72   3   1
 109   9]
 [  0  31  33   0  98   3   2 165  10 188  36 110  40   7   0 143   2   5
  78  12]
 [  0  14  32   1  29   0   9   0  27   1   9 589   6   0   0 120   1   0
 142   8]
 [  2  55  92   1 232  18   3 113  13  77  61  45  56  31   0  91  19  11
  42  13]
 [126  42 235   1 155   5   0  13  22   5  53  16  43  10  15  25  32  61
  30 101]
 [ 75  57 310  10 104   8   1  13  23   2  23  13  32  20   2  23  97  26
  64  93]
 [  2   6  48   0  85 391   0   0   4   0  10   2  13 253   0   4 144   7
   3  22]
 [  0   5  11   1  14 743   0   0   3   0  13   0   6 145   1   1  50   0
   2   4]
 [ 14  10  71   0  32   2 662   0  22   5  18  34   0   2  28  17   1  23
  33  17]
 [ 18 115 104   4 128  13  11  19  52  50  24 108  21  17   1 135  17  16
  92  39]
 [100  64 111   9 205   4   0   0  37   1  11  27  21   7  46  48  14 158
  34  93]
 [ 15  18  54   2  72   4   0   2 677   1   3  22   8   4  15  21   7  31
  10  21]
 [  9  12  44 784  41   0   0   1   7   1  13  10   3   3  21  15   0  22
   5   6]
 [198   2 120   1  40   1   6   0   7   0  21   4   9   1 317   2   7 126
   5  43]
 [ 68  10   8   7  45   1   0   0   0   0  20   2  10   1 607   4   1 132
   2  22]
 [186   7  80  10  36   0   3   0  27   0  71   1   7   7 187   5   2 107
   6  33]
 [ 62   5  59 281  44   4   1   0   5   0  24   0   7   1  47   8   1  40
   5  34]]
```

Apply log transformation after NMF



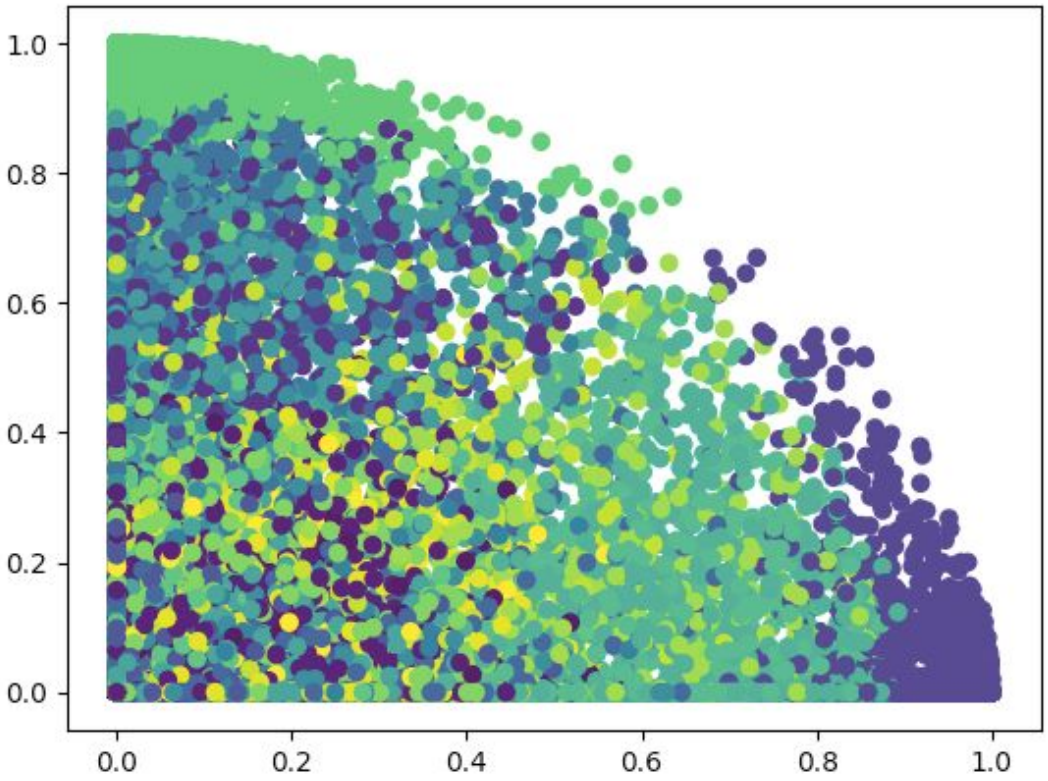
homogeneity score:
0.31493004135845737
completeness score:
0.3509784529330501
V-measure:
0.3319785215106319
adjusted rand score:
0.12536932676272894
adjusted mutual info score:
0.3127064830352485

Contingency matrix:



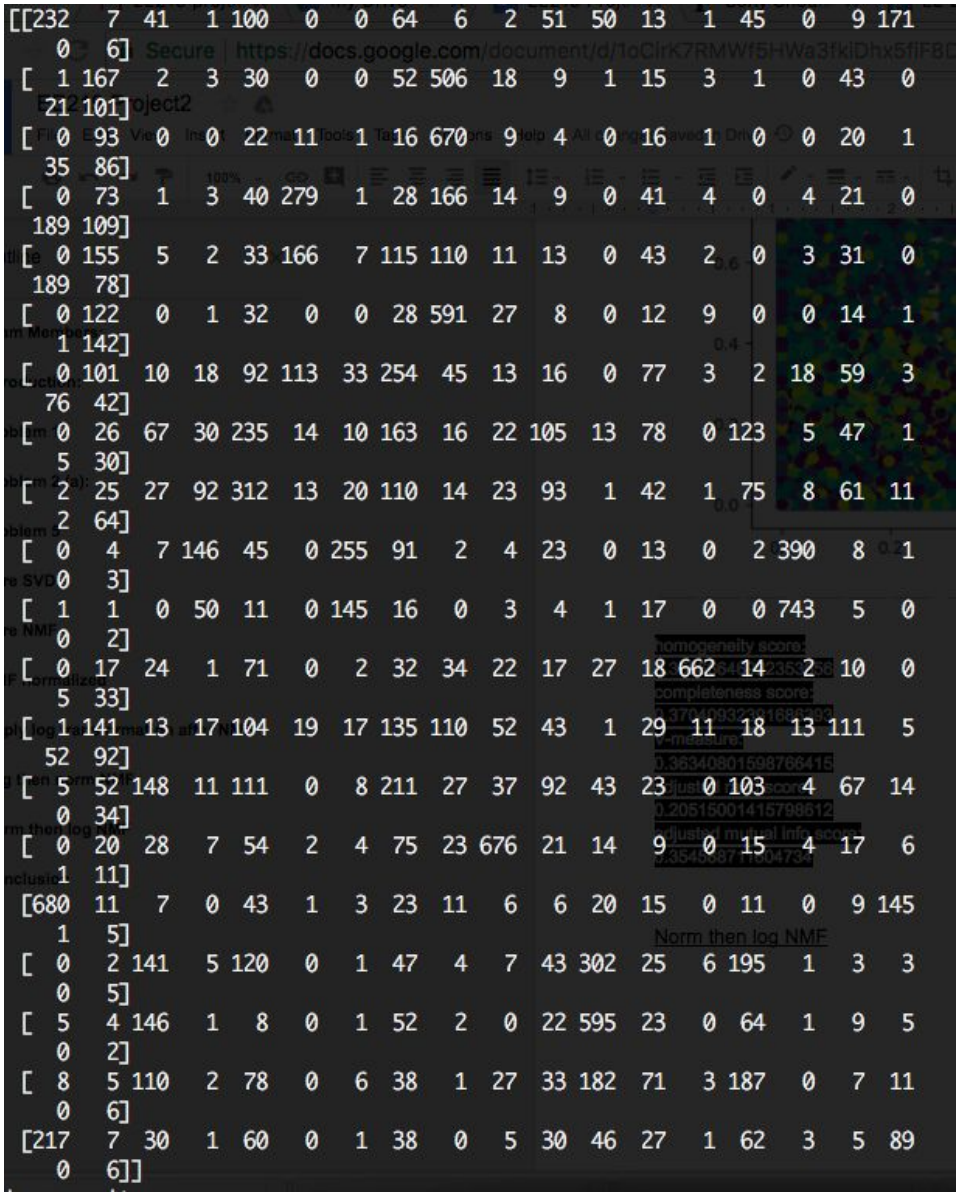
```
[[  0  79  85   0   0   0  51 241   0   1   2 139   0   8  52   0   1   4
    1 135]
 [  1  18  74   2  82   0 239   1   0  14   9 103   0   0   0   5   0 422
    0  3]
 [  0   8  44   0 327   6 122   0   0   3   8  36   0   0   0  21   0 408
    0   2]
 [  2  26  81   4  39 112 157   0   0   7  26  62  50   0   0 188   0 221
    7   0]
 [  1  16  70  12   7  54 274   0   0   8  21 190   1   0   0 155   0 149
    3   2]
 [  9  19  72   2 107   0 204   0   0  25   4  61   0   0   0   1   0 484
    0   0]
 [  1  28 113  23   8  36 229   1   3   7  30 322   1   0   0  94   0  57
   18   4]
 [  0  85 341  18   0   0 139   0   0   8  27 252   0   0   0   7   0  10
    7  96]
 [  0 181 363   4   0   0 161   8   0  16  13 169   0   0   0   7   0  13
   23  38]
 [  0  36 105   3   0   0  50   0 125   2   6 228   0   0   0   0   0   2
  432   5]
 [  0   9  22   6   0   0  21   1 405   3   7  69   0   0   0   0   0   0
  454   2]
 [411  51  59   7   7   0  58   0   0  20   9  52   0   8   0   1 239  35
    2  32]
 [  6  27 198   4   5   2 375   0   0  32   8 191   0   0   0  22   0  95
   12   7]
 [  0  28 216   2   2   0 252   6   0  18   7 300   0   3   1   0   0   6
    3 146]
 [  0  20 104   0   0   0 150   0   0 511   2 137   0   1   0   2   0  14
    4  42]
 [  0   8  59   0   1   0  80 485   0   3   5  81   0   2 222   1   0   5
    0  45]
 [  4 125 124   3   1   0  16   1   0   3  10  86   0 166   0   0   2   0
    0 369]
 [  0  10  41   0   0   0  25   3   0   0  19 105   0 368   2   0   0   1
    2 364]
 [  1  65 108  13   0   0  27   5   0  14  50  87   0  80   2   0   1   0
    2 320]
 [  1  36  81   2   0   0  43 173   0   4  16  91   0  15  71   0   0   1
    0  94]]
```

Log then norm NMF

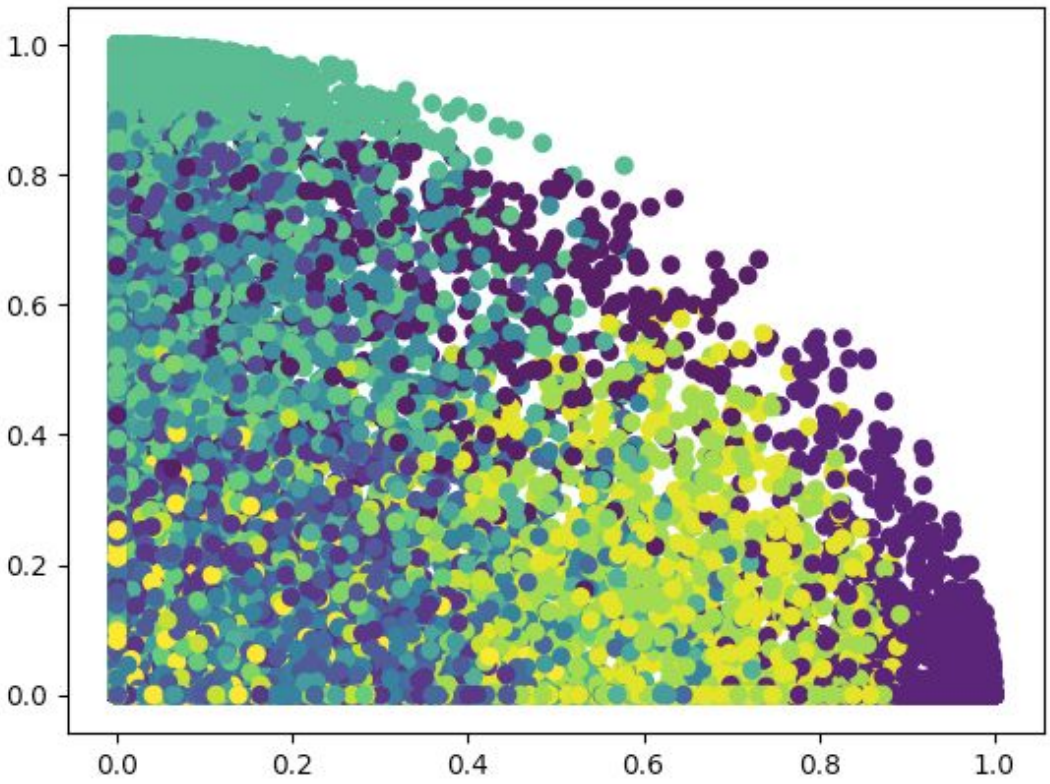


homogeneity score:
0.35666646922353656
completeness score:
0.37040932391686393
V-measure:
0.36340801598766415
adjusted rand score:
0.20515001415798612
adjusted mutual info score:
0.354588711604734

Contingency matrix:

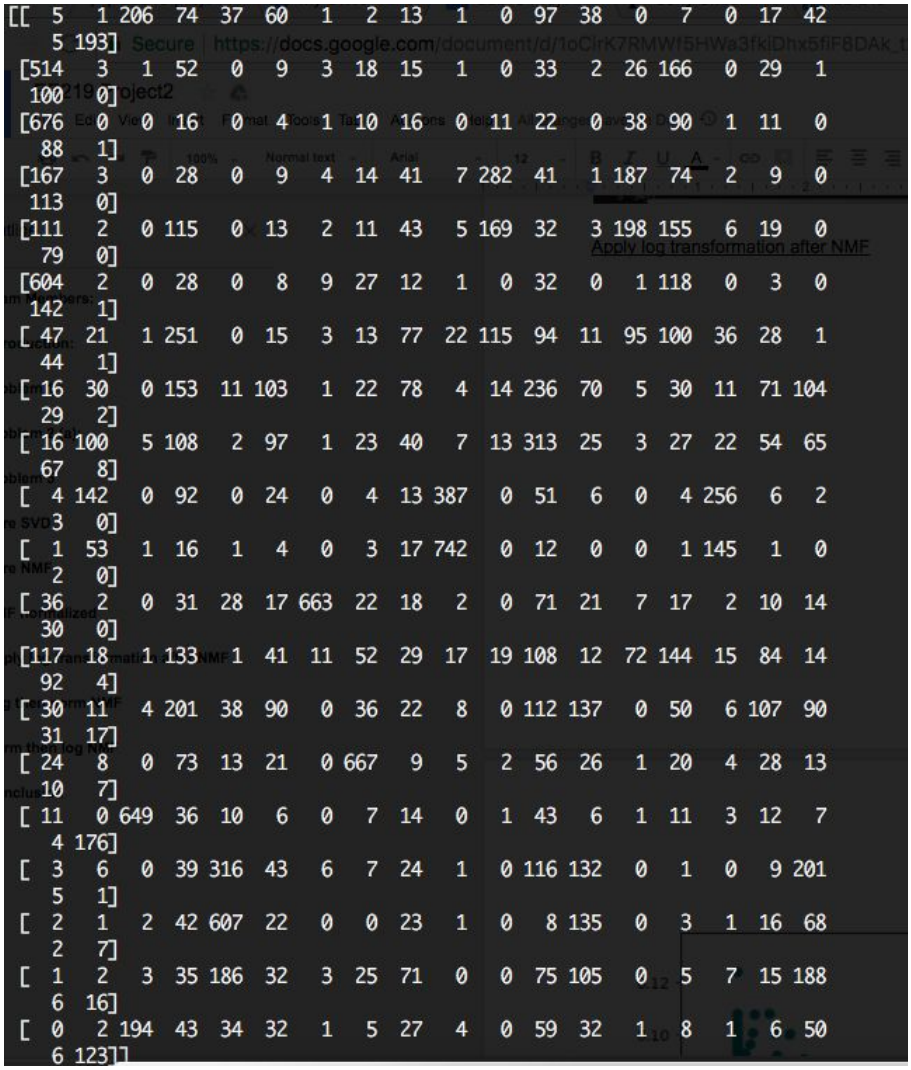


Norm then log NMF



homogeneity score:
0.35459294749536646
completeness score:
0.367628357927911
V-measure:
0.3609930143064373
adjusted rand score:
0.20782748498930934
adjusted mutual info score:
0.35250868583233697

Contingency matrix:



As one can observe from the measures for different transformations for NMF, we can see that the measures are similar, around 0.2,0.3, and log transformation after NMF is performing slightly worse compare to others (since it has 0.125 for adjusted rand score)

Conclusion

In this project, we find the proper representations of the data, perform K-means clustering and evaluate the performance. Moreover, we try different preprocess methods.

Specifically, we build the TF-IDF matrix, apply K-means clustering with k = 2 using TF-IDF data, try the five different measures for evaluations, did dimensionality reduction

with both SVD and NMF. Moreover, we visualized the clustering, tried transformations like log transformation. Lastly, we expand the dataset to 20 clusters and evaluate with different dimensions and transformations.