EE219 PROJECT\_2 REPORT

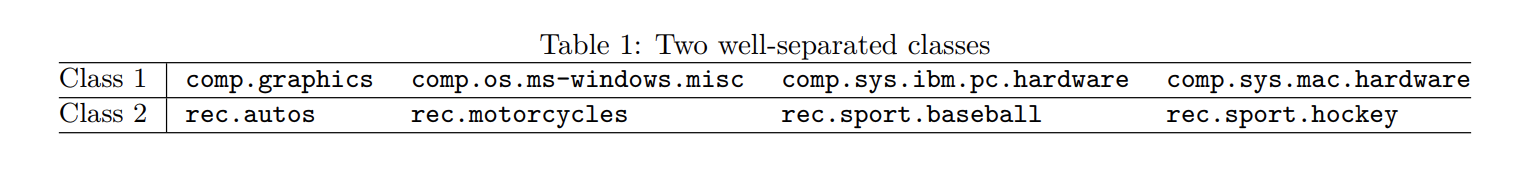
Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them.

Given a set of data points, we can use a clustering algorithm to classify each data point into a specific group. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features. Clustering is a method of unsupervised learning and is a common technique for statistical data analysis used in many fields.

Before we get started we set the random seed to 42 globally.

We work with “20 Newsgroups” dataset available in sklearn library. It is a collection of approximately 20,000 documents, partitioned (nearly) evenly across 20 different newsgroups, each corresponding to a different category (topic). Each topic can be viewed as a “class”. In order to define the clustering task, we pretend as if the class labels are not available and aim to find groupings of the documents, where documents in each group are more similar to each other than to those in other groups. We then use class labels as the ground truth to evaluate the performance of the clustering task.

We load all the data belonging to the dataset and partition them in two categories as shown below:



We perform K-means clustering on it and see if we can retrieve the known classes.

We would like to evaluate how purely the a priori known classes can be reconstructed through clustering. That is, we take all the documents belonging to these two classes and perform unsupervised clustering into two clusters. Then we determine how pure each cluster is when we look at the labels of the documents belonging to each cluster.

QUESTION 1: Report the dimensions of the TF-IDF matrix you get.

After loading the data, we have to create the TF-IDF representation of it so that it is easy to do analysis on.  **Tf**-**idf** stands for term frequency-inverse document frequency, and is computed using the CountVectorizer() function with min\_df=3, followed by the TfidfTransformer() function. The stop words were eliminated as defined in NLTK and no stemming or lemmatization was performed.

The resultant matrix was of the dimensions:

(7882, 27768)

(7882, 27768)

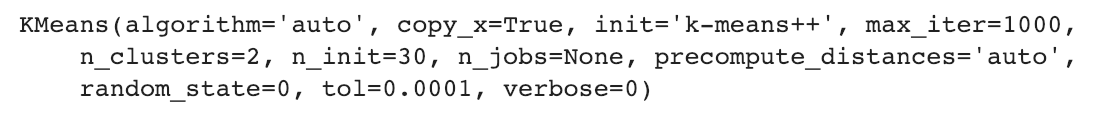
K-means clustering is a simple and popular clustering algorithm. To begin, we first select a number of classes/groups to use which in this case was 2, since there were 2 classes we had divided the data into and randomly initialize their respective center points. We use the tf-idf data we computed to perform K-means clustering.

Given a set of data points {x1, . . . , xN } in multidimensional space, it tries to find 2 clusters such that each data point belongs to one and only one cluster, and the sum of the squares of the distances between each data point and the center of the cluster it belongs to is minimized.

1. Each data point is classified by computing the distance between that point and each group center, and then classifying the point to be in the group whose center is closest to it.
2. Based on these classified points, we recompute the group center by taking the mean of all the vectors in the group.
3. Repeat these steps for a set number of iterations or until the group centers don’t change much between iterations. You can also opt to randomly initialize the group centers a few times, and then select the run that looks like it provided the best results.

In this project we use the scikitlearn library’s function KMeans() with the following parameters: random\_state=0, max\_iter =1000 and n\_init = 30.

It creates the following model:

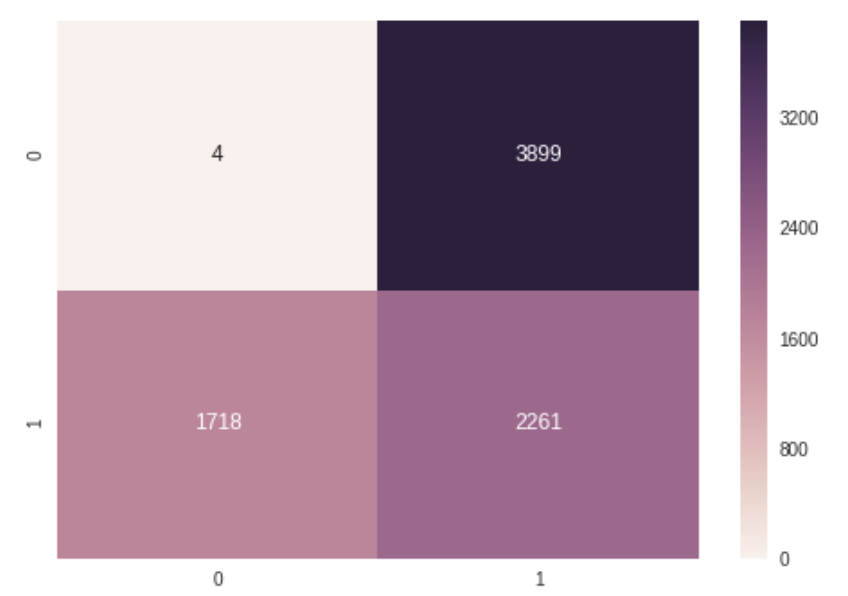


Using this model we use the kmeans\_labels\_ attribute to get all the predicted labels from the model. Following this, we compare the predicted kmaens\_labels with the original labels as ground truth.

QUESTION 2: Report the contingency table of your clustering result.

To represent this result, we compute the contingency table A, which is the matrix whose entries Aij is the number of data points that belong to both the class Ci the cluster Kj .

The Contingency Matrix which is a function of the metric package of scikitlearn is used to compute the contingency matrix and Seaborn is used to Visualize it.



QUESTION 3: Report the 5 measures above for the K-means clustering results you get.

Apart from the contingency matrix. There are multiple other metrics to gauge the performance of the KMeans algorithm, for a given partition of the data points with respect to the ground truth.

We use the measures homogeneity score, completeness score, V-measure, adjusted Rand score and adjusted mutual info score, all of which can be calculated by the corresponding functions provided in sklearn.metrics.

**Homogeneity** is a measure of how “pure” the clusters are. If each cluster contains only data points from a single class, the homogeneity is satisfied. This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won’t change the score value in any way.

Using the homogeniety\_score() function on the actual and predicted labels, we get a value of:

homogeneity:   0.2535958928926043  
  
**Completeness** clustering result satisfies completeness if all data points of a class are assigned to the same cluster. Both of these scores span between 0 and 1; where 1 stands for perfect clustering.

Using the completeness\_score() function on the actual and predicted labels, we get a value of:

completeness:  0.334815748824373

**V-measure** is defined to be the harmonic average of homogeneity score and completeness score.

Using the v\_measure\_score() function on the actual and predicted labels, we get a value of:

v\_measure : 0.28860033608397917

**Adjusted Rand Index** is similar to accuracy measure, which computes similarity between the clustering labels and ground truth labels. This method counts all pairs of points that both fall either in the same cluster and the same class or in different clusters and different classes.

Using the  adjusted\_rand\_score() function on the actual and predicted labels, we get a value of:

adjusted\_rand\_score:  0.18076179588914554

**Adjusted mutual information score** measures the mutual information between the cluster label distribution and the ground truth label distributions.

Using the  adjusted\_mutual\_info\_scoree() function on the actual and predicted labels, we get a value of:

adjusted\_mutual\_info\_score:  0.25352755133060884

We see that the high dimensional sparse TF-IDF vectors do not yield a good clustering result. One of the reasons is that in a high-dimensional space, the Euclidean distance is not a good metric anymore, in the sense that the distances between data points tends to be almost the same. Since its objective is to minimize the sum of within-cluster l2 distances, it implicitly assumes that the clusters are isotropically shaped, i.e. round-shaped. When the clusters are not round-shaped, K-means may fail to identify the clusters properly. Even when the clusters are round, K-means algorithm may also fail when the clusters have unequal variances.

Thus we try to find a “better” representation tailored to the way that K-means clustering algorithm works, by reducing the dimension of our data before clustering. We use Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) for dimensionality reduction.

**Singular Value Decomposition**

Singular Value decomposition is used to factorize a real or complex matrix into the form UΣV form, where U is an mxm unitary matrix, Σ is a rectangular diagonal matrix with non-negative real numbers on the diagonal, and V {\displaystyle \mathbf {V} }VV v is an {\displaystyle n\times n} real or complex unitary matrix.

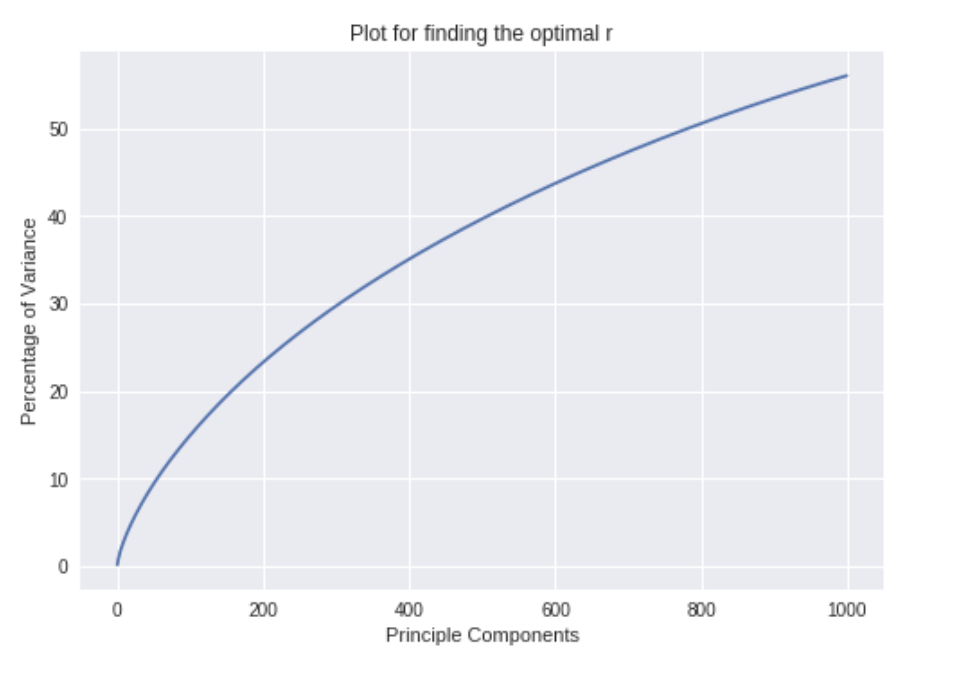
{\displaystyle \mathbf {M} }The diagonal entries {\displaystyle \sigma \_{i}} of {\displaystyle \mathbf {\Sigma } } are known as the singular values of the matrix.

We find the effective dimension of the data through inspection of the top singular values of the TF-IDF matrix and see how many of them are significant in reconstructing the matrix with the truncated SVD representation. We calculate the Singular Value Decomposition with 1000 components, on the tf-idf data and calculate the variation ratios of each component using the explained\_variance\_ratio attribute of the truncatedSVD() function model.

QUESTION 4: Report the plot of the percent of variance the top r principle components can retain v.s. r, for r = 1 to 1000.

To find the percentage of variance of the top r principle components, we compute the cumulative variation ratio for r components and plot that against the number of components. Plotting the graph shows us a gradual increase in the percentage of variance, since more components add more variance to the data.

The plotted graph looks as follows:

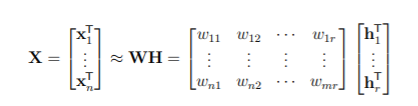


**Non-negative Matrix Factorization (NMF)**

The main goal of NMF is to approximate our data matrix with two non-negative matrices . In our case, n is the number of documents, m is the term, and r is the reduced dimension. Our reduced matrix that we will use, is the W we found by minimizing the loss function below:



The intuition of W and H is that we are trying to explain X, as a non-negative linear combination of the r topics.



Each hiT represents an importance score of each term in each topic. After solving the optimization problem above, we get our reduced dimension using NMF which is our W.

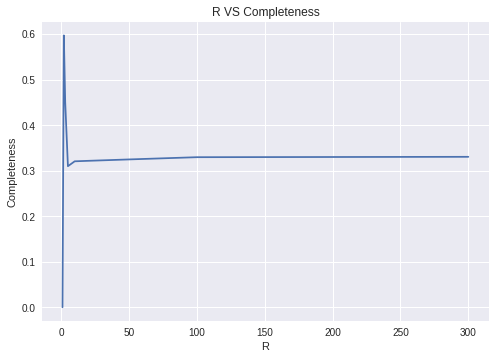
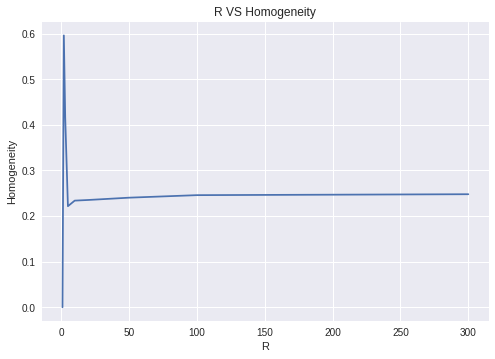
QUESTION 5: Let r be the dimension that we want to reduce the data to (i.e. n\_components).  
Try r = 1,2,3,5,10,20,50,100,300, and plot the 5 measure scores v.s. r for both SVD and NMF.  
Report the best r choice for SVD and NMF respectively.

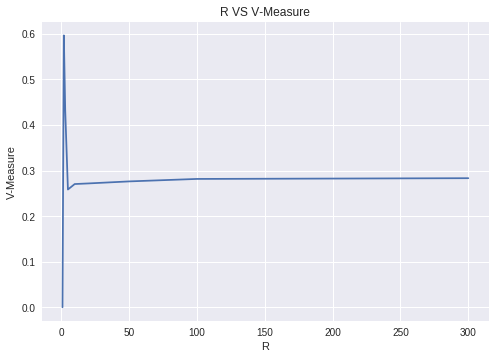
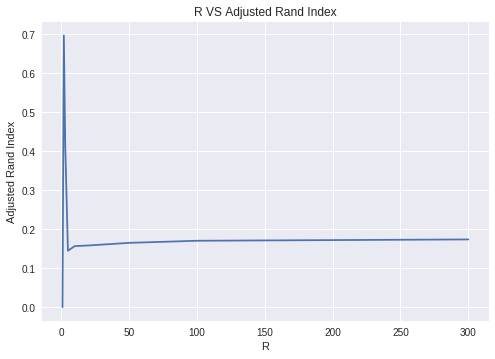
In this question, we use both SVD and NMF by selecting different number of components and comparing them so that we can choose the number of components that give the best clustering result. We first start with SVD.

We do not have to explicitly compute SVD again every time, for each components. This is because we have already computed SVD for 1000 components in our earlier problem. We just have to select the columns from the SVD matrix based on the number of components we want. For example: If we want 100 components, we can choose the first 100 columns of the SVD matrix

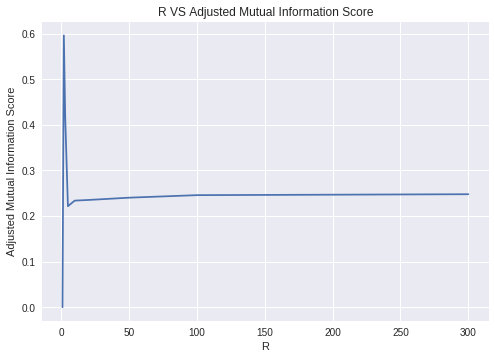
that we computed earlier. The reason we can directly select these 100 columns without having to compute SVD because with r=1000 in our past problem, we have already got the data that is projected onto the 1000 components. We can just choose the components that we want to keep. After performing SVD with the specified number of components, we perform K-means clustering and evaluate with the 5 metrics we did with the previous question giving us the results below:

**Homogeneity**

**Completeness**

** V-Measure Adjusted Rand Index**

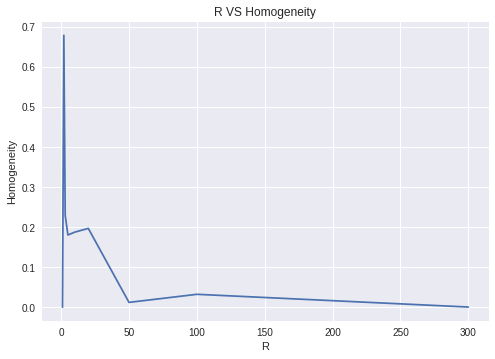
**Adjusted Mutual Information Score**

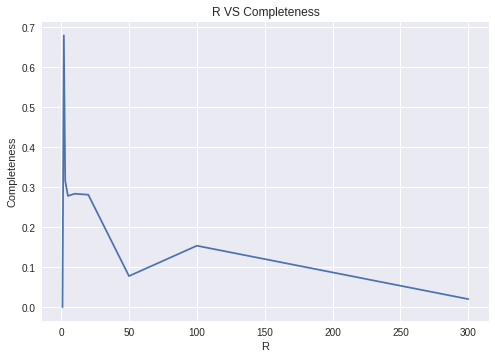
****

From the graphs, we can see that the best value of r or number of components is 2 for all evaluation matrices. Next, we will use NMF and see how the results are.

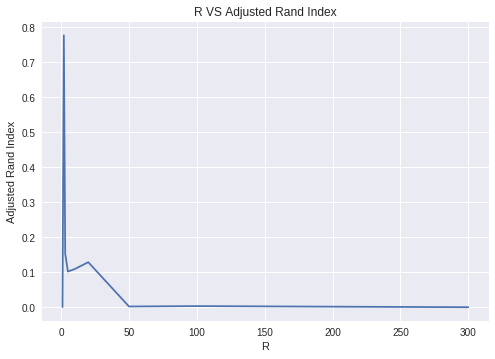
With NMF, we cannot do what we did for SVD, where we can just choose the top r components from a computed larger NMF matrix. This is because the value of W differs for each number of components that we would like to choose. In this case, we would have to call the compute NMF function for each number of components we would like to compute for. This can take several minutes for large number of components such as r=300.

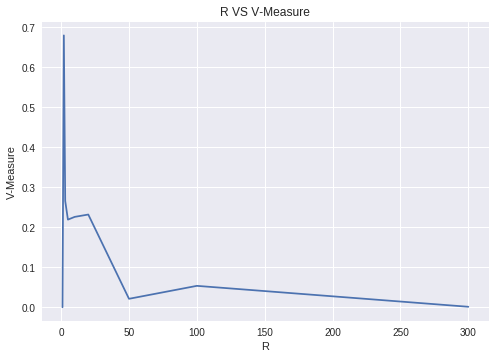
With NMF we get the results as below:

** Homogeneity Completeness**

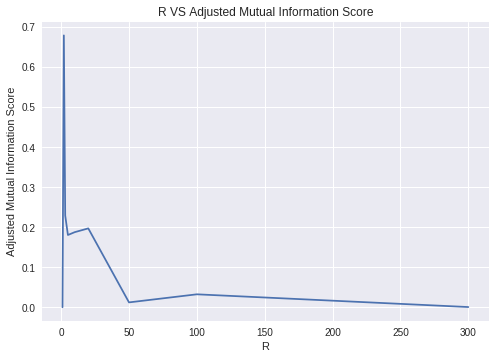
****

**V-Measure Adjusted Rand Index**

****

****

**Adjusted Mutual Information Score**

****

Again, we see that the best value for NMF is r = 2.

This is why we do not run into a problem of having to choose which r we want to choose based on the evaluation metrics. However, if we are faced with that problem, we will have to weigh which metrics we would care about more. For instance, homogeneity might not be a good evaluation metrics. This is because if we do not know the number of clusters, and we assigned each point to its own cluster, we can achieve a homogeneity of 1 which is perfect clustering. However, this is a very bad clustering. We have reported the values of these scores with the best r on SVD and NMF below.

**SVD r=2 metrics value:**

Homogeneity - 0.5960249753578137

Completeness - 0.5971396831400697

V-Measure - 0.5965818085440118

Adjusted rand index - 0.6972989204562027

Adjusted Mutual Score Information - 0.5959879913066645

**NMF r=2 metrics value:**

Homogeneity - 0.6790483562300399

Completeness - 0.680131609210451

V-Measure - 0.6795895510492934

Adjusted rand index - 0.7770177788377391

Adjusted Mutual Score Information - 0.6790189730005426

From the scores, we can see that NMF does better in all categories as compared to SVD. This could be because our data is text data. In SVD, when you decomposed the matrix, you get some positive and some negative values. However, in NMF, you get all positive values. Here, we are decomposing our TF-IDF matrix which are all positive values since we are counting the terms in the documents. This is why having all positive values in our decomposition makes more sense causing NMF to give a better result than SVD.

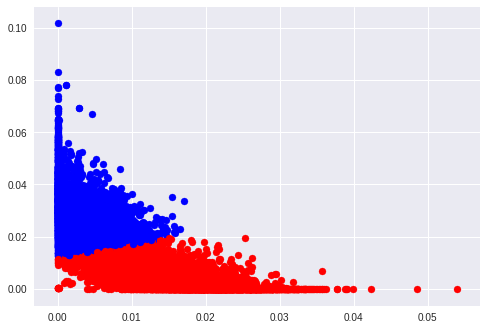
QUESTION 6: How do you explain the non-monotonic behavior of the measures as r increases?

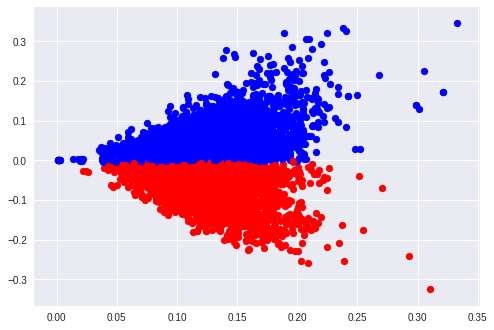
Intuitively, we should be performing better as we increase the number of components we choose because we are capturing more information in X which is our data. However,

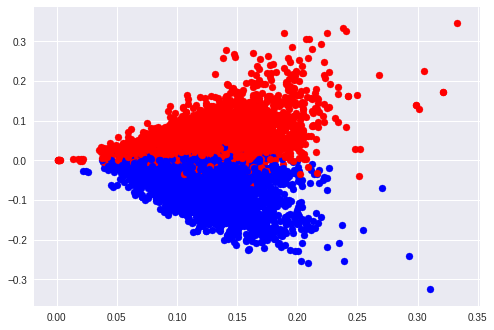
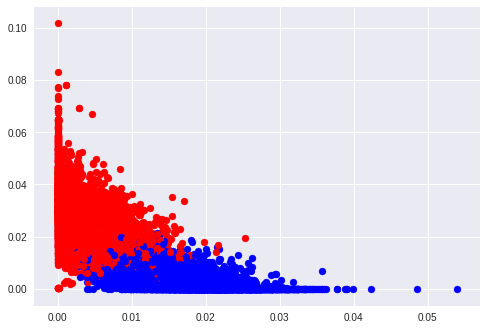
K-means uses distance in order to classify which points belong to which cluster. This is what causes the problem when we have more components, since the dimension increases and distance measure starts to not make sense in higher dimension. This is known as the **curse of dimensionality**. Imagine that your data points are on a line ranging from [0 1]. Assume that your data is uniformly distributed. If we divide that interval into 10 cells, then we would likely have our data in every cell. However, if we move this into the second dimension keeping our unit of discretization to the same which is 0.1, we would now have a 100 two-dimensional cell. In this case, we can see that most of the cells would be empty. If we raise this to an even higher dimension, almost all of the cells will be empty. Intuitively, our data becomes lost in space when we move to higher dimensions and the distance measure between the points remain the same which is why there is a non-monotonic behavior when we decide to choose more components. Choosing more components doesn’t make the clustering better.

QUESTION 7: Visualize the clustering results for: • SVD with its best r. • NMF with its best r

Using the best r (r = 2 for both SVD and NMF), we plotted the clustering results as shown below:

****  
 **SVD Clustering r = 2 NMF Clustering r= 2**



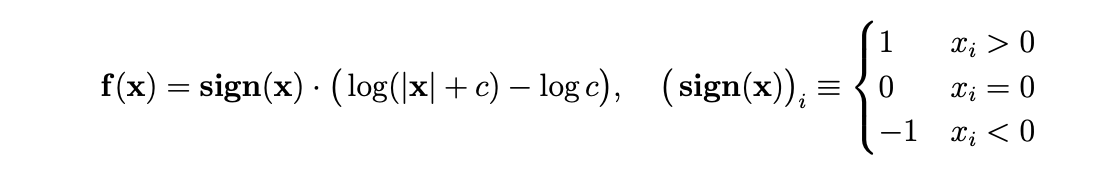
 **Ground Truth SVD r = 2 Ground Truth NMF r = 2**

We can see that the results from the ground truth and the results from our K-means clustering algorithm looks very similar. However, the colors are swapped. This means that in the ground truth, when the values are 1, our clustering algorithm predicts it as 0. However, this is not an issue and can be expected because when clustering, we do not know the ground truth label. When evaluating the results, we are just matching the most likely cluster to the ground truth cluster. In this case, the blue from our k-means clustering will be mapped to the red of the ground truth because they have more similarities.

Now we try the transformation methods below to see whether they increase the clustering performance. We perform transformation on SVD-reduced data and NMF-reduced data, respectively. We use the best r we had in previous parts namely, r = 2.

Scaling features such that each feature has unit variance, i.e. each column of the reduced-dimensional data matrix has unit variance (if we use the convention that rows correspond to documents). (Centering the data such that mean = 0 is not needed since the relative distances between the points is not going to change).

Applying a non-linear transformation to the data vectors. Here we use logarithm transformation below as an example (with c = 0.01)

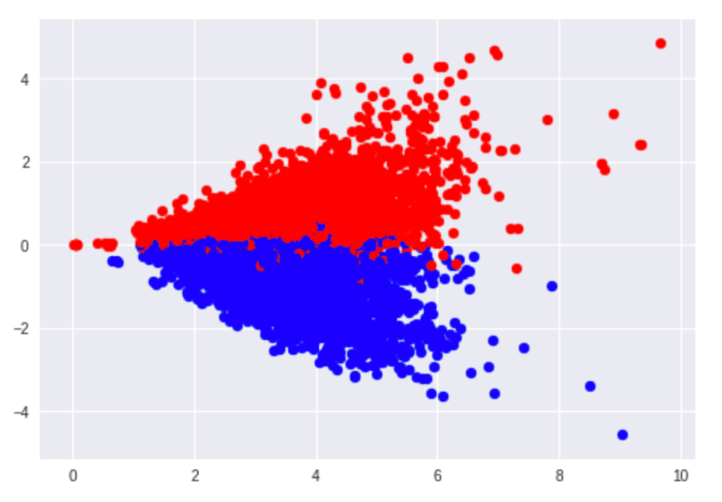


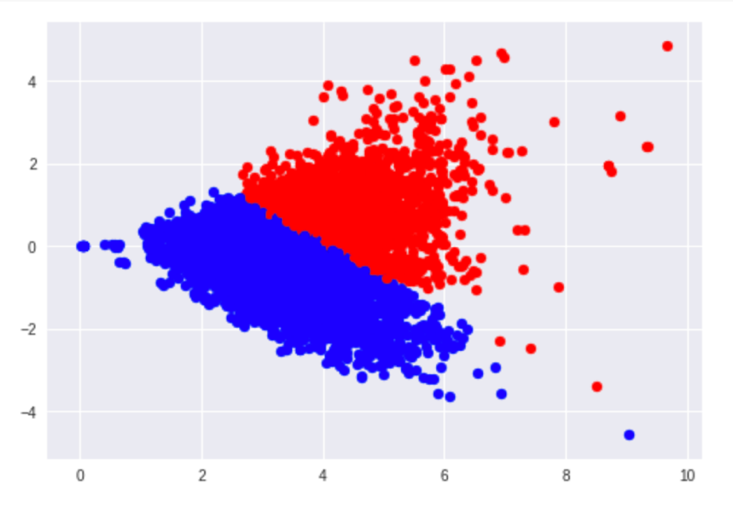
We then combine both transformations (in both orders). We end up trying 2 × (2 + 2) = 8 combinations.

QUESTION 8: Visualize the transformed data as in part (a)

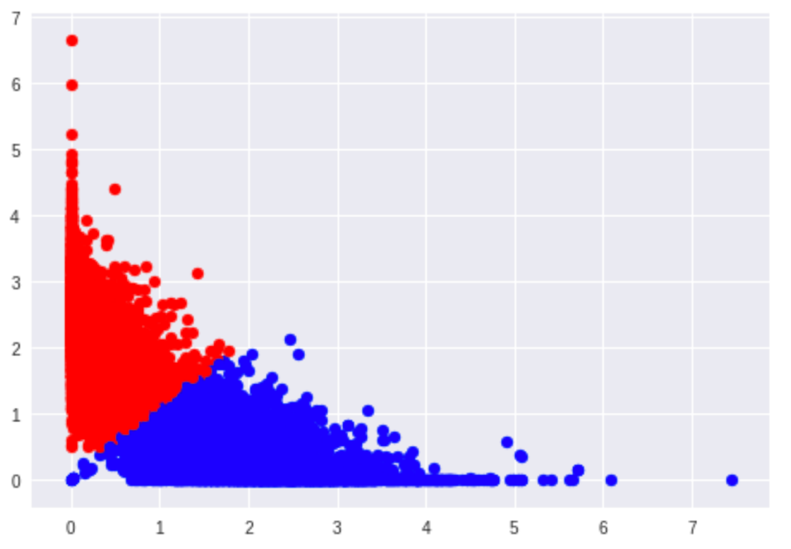
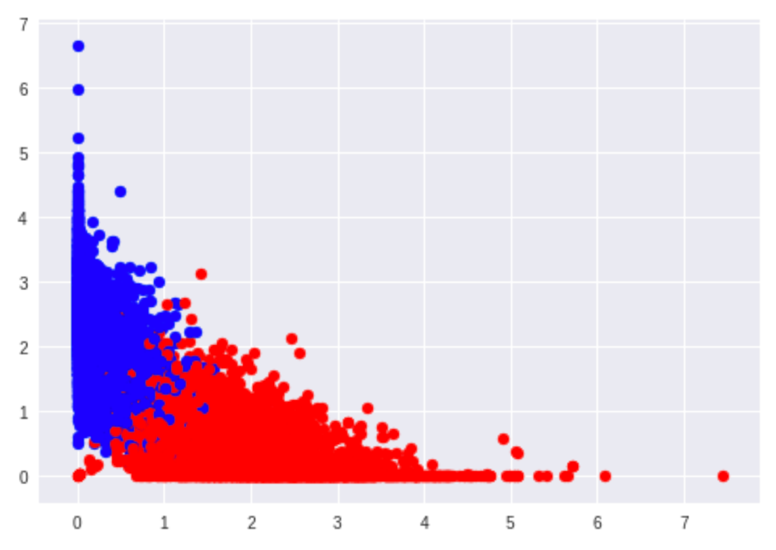
Using the best r (r = 2 for both SVD and NMF), we plotted the clustering results for both transformations and all combinations:

**SVD WITH NORMALIZATION (K-Means and Ground Truth)**

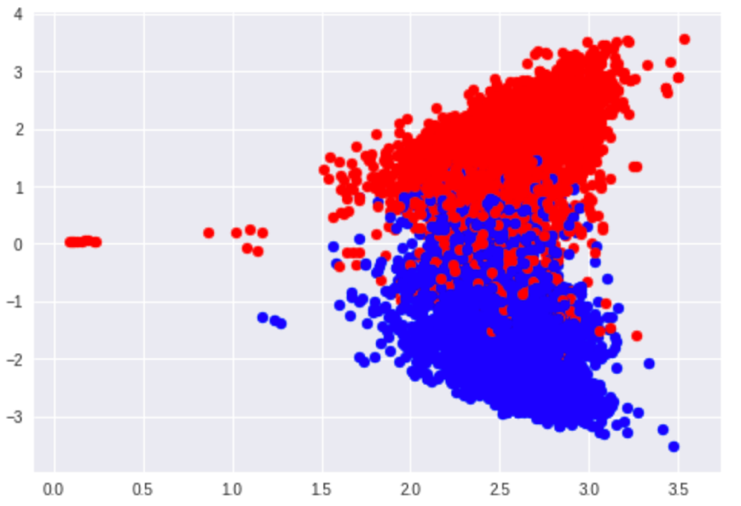


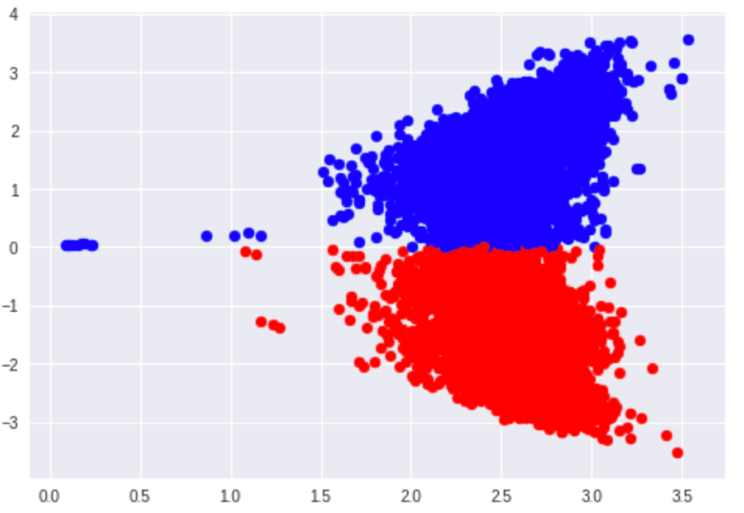


**NMF WITH NORMALIZATION (K-Means and Ground Truth)**

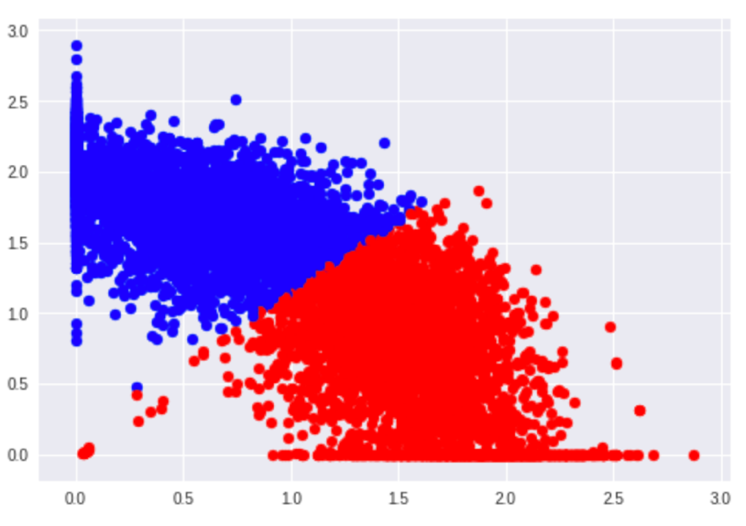
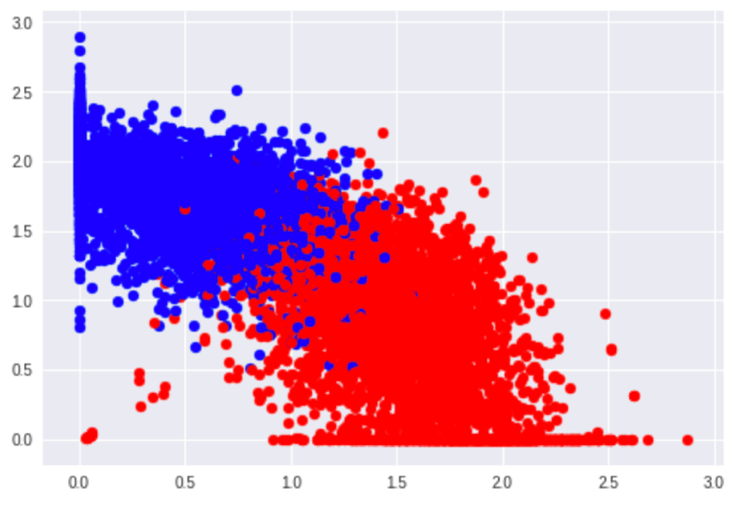
****

**SVD WITH LOGARITHMIC TRANSFORMATION (K-Means and Ground Truth)**

****

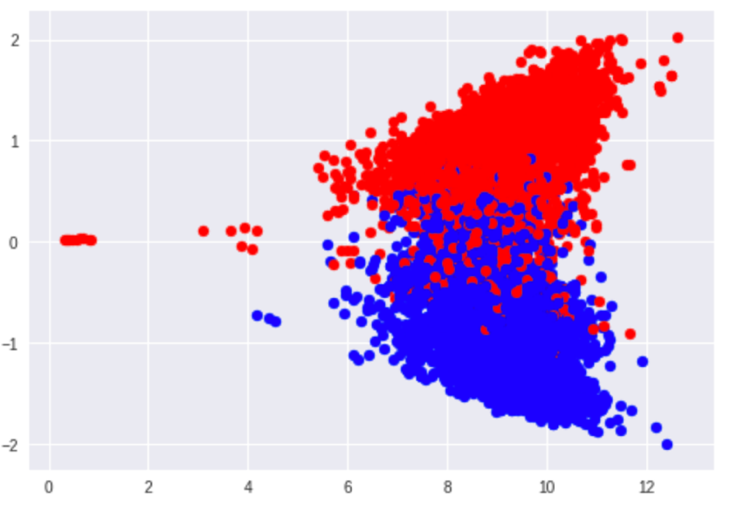
****

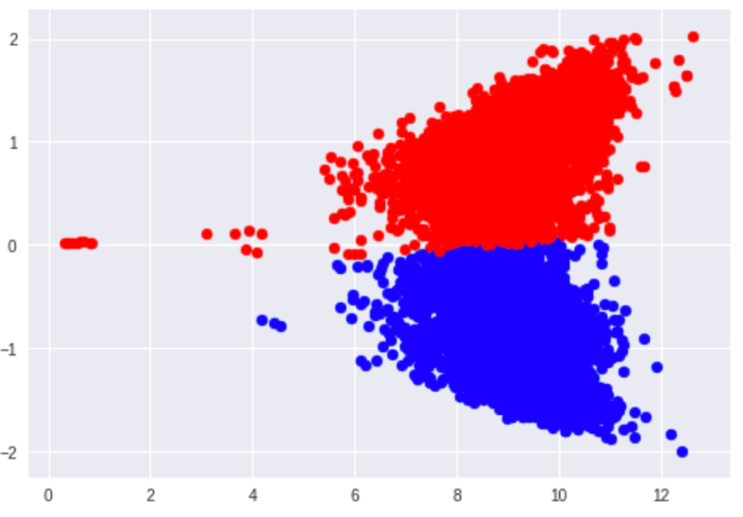
**NMF WITH LOGARITHMIC TRANSFORMATION (KMeans and Ground Truth)**

****

**SVD WITH LOGARITHMIC TRANSFORMATION THEN NORMALIZATION**

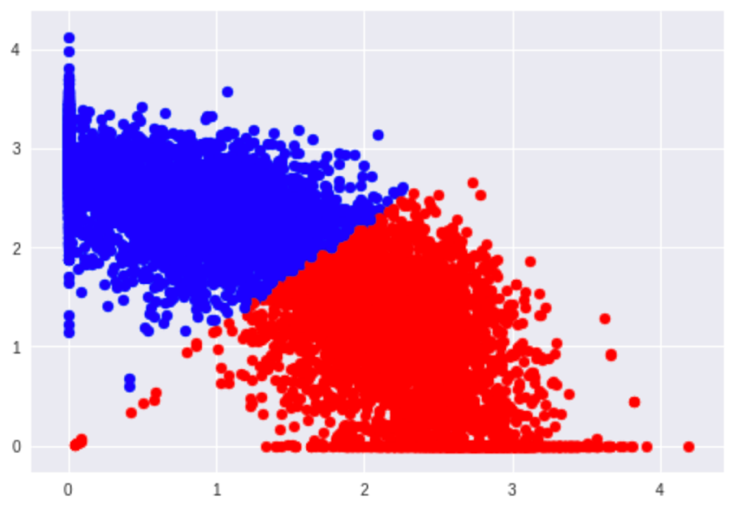
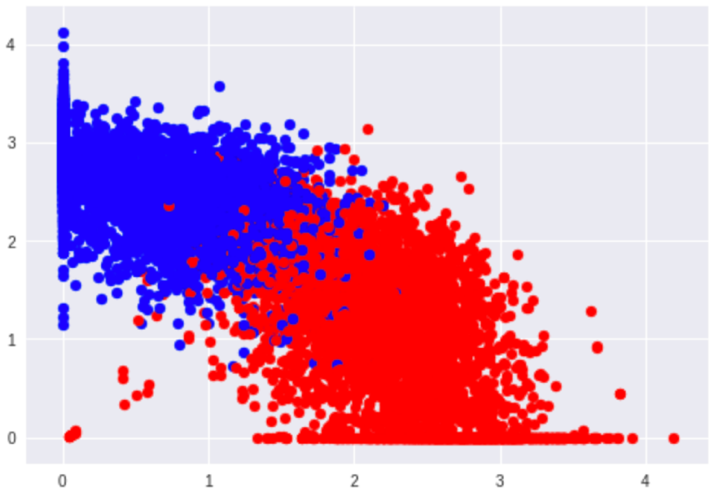
**( K- Means and Ground Truth)**

****

****

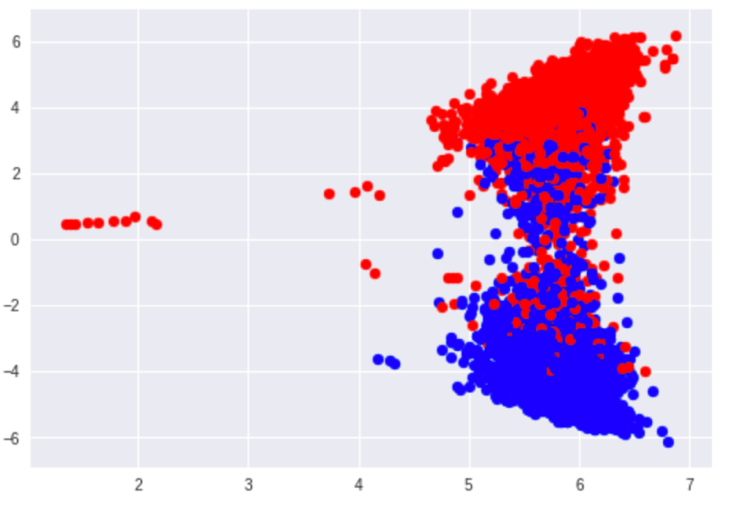
**NMF WITH LOGARITHMIC TRANSFORMATION THEN NORMALIZATION**

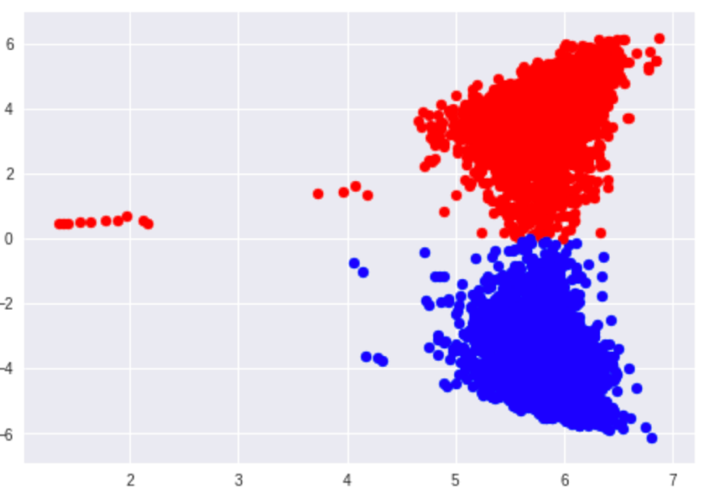
**( K- Means and Ground Truth)**

****

**SVD WITH NORMALIZATION THEN LOGARITHMIC TRANSFORMATION**

**( K-Means and Ground Truth)**

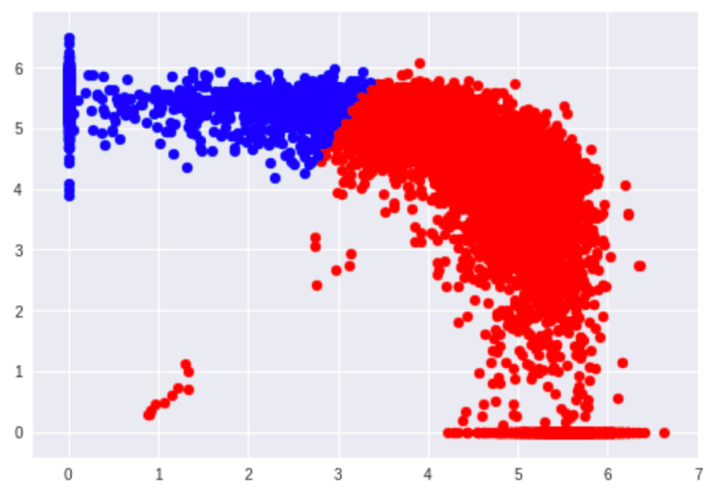
****

****

**NMF WITH NORMALIZATION THEN LOGARITHMIC TRANSFORMATION**

**( K- Means and Ground Truth)**

****

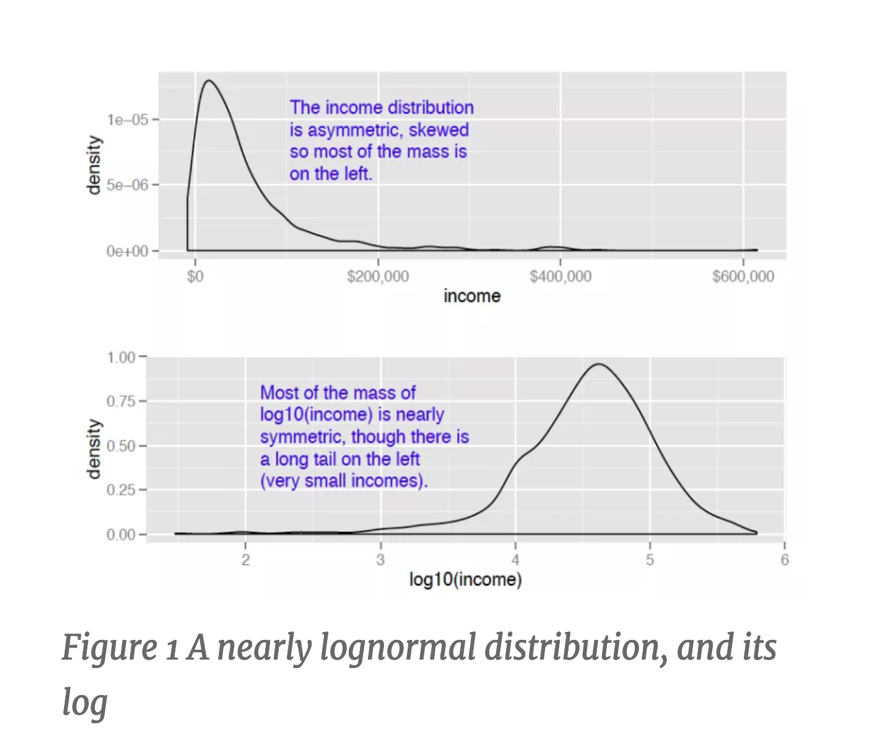
****

**QUESTION 9**: Can you justify why the “logarithm transformation” may improve the clustering results?

The data transformation we apply depends on the kind of modeling method used. Depending on the underlying assumptions made by the modeling method the data may have to be transformed to satisfy these assumptions to ensure the best outcome from the modeling [1].

For example, the K-Means clustering algorithm uses the Euclidean distance to assess similarity between data points and assign them to clusters. In doing so, there is an implicit isotropic assumption of feature distributions (Isotropy is uniformity in all orientations). This assumption is wrong especially if some features are skewed compared to the others.

Logarithmic transformations on data restore symmetry to it as illustrated by the following graph:



Also, log transforming data with values that range over several orders of magnitude is also a good idea because modeling techniques have a difficult time with very wide data ranges. Such data often come from multiplicative processes so log units are more natural in such cases.Taking the logarithm only works if the data is non-negative. Hence, we applied the signed logarithmic function we defined above. This signed logarithm function takes the logarithm of the absolute value of and multiplies by the appropriate sign.

**QUESTION 10**: Report the new clustering measures (except for the contingency matrix) for the clustering results of the transformed data.

**Normalized Svd Resutls**

homogeneity: 0.23609131805072042

completeness: 0.26450291273288246

v\_measure : 0.24949085488903866

adjusted\_rand\_score: 0.2556510317794412

adjusted\_mutual\_info\_score: 0.2360213789796711

**Normalized Nmf Resutls**

homogeneity: 0.6828038321574016

completeness: 0.6856459752144646

v\_measure : 0.6842219522524521

adjusted\_rand\_score: 0.7734426774605906

adjusted\_mutual\_info\_score: 0.6827747927166996

**Logarithmically Transformed Svd Resutls**

homogeneity: 0.6103154102550904

completeness: 0.6102847108358388

v\_measure : 0.6103000601594027

adjusted\_rand\_score: 0.7173615346457451

adjusted\_mutual\_info\_score: 0.6102490340872908

**Logarithmically Transformed Nmf Resutls**

homogeneity: 0.6757033916294439

completeness: 0.6791388923298461

v\_measure : 0.6774167862456966

adjusted\_rand\_score: 0.7649847921825073

adjusted\_mutual\_info\_score: 0.6756737021308169

**Logarithmically Transformed Then Normalized Svd Resutls**

homogeneity: 0.6095947774726546

completeness: 0.6095541488812544

v\_measure : 0.609574462499973

adjusted\_rand\_score: 0.7165020095866556

adjusted\_mutual\_info\_score: 0.6095184058375314

**Logarithmically Transformed Then Normalized Nmf Resutls**

homogeneity: 0.6863512458670784

completeness: 0.6890551449829493

v\_measure : 0.6877005376394758

adjusted\_rand\_score: 0.777017803240186

adjusted\_mutual\_info\_score: 0.686322531195829

**Normalized Then Logarithmically Transformed Svd Resutls**

homogeneity: 0.6094447324650026

completeness: 0.609408642445872

v\_measure : 0.609426686921128

adjusted\_rand\_score: 0.7165020096856823

adjusted\_mutual\_info\_score: 0.6093728858160544

**Normalized Then Logarithmically Transformed Nmf Resutls**

homogeneity: 0.31316451556199965

completeness: 0.38284164004856497

v\_measure : 0.3445153919295321

adjusted\_rand\_score: 0.2487703014872082

adjusted\_mutual\_info\_score: 0.3131016304947138

SVD factors contains both positive and negative entries while NMF factors are strictly positive. This is useful in case of text mining where usually TF or TF-IDF matrix is being factorized. Since entries of original matrix are positive and has physical meaning (term frequencies...), once would want factors to be positive so that physical connections are be directly made [2].

When normalization is done after the log transformation, NMF does much better because it is a true positive representation of the tf-idf data as explained above.

In the simple normalization case and in the simple log transformation case, NMF does better than SVD as expected.

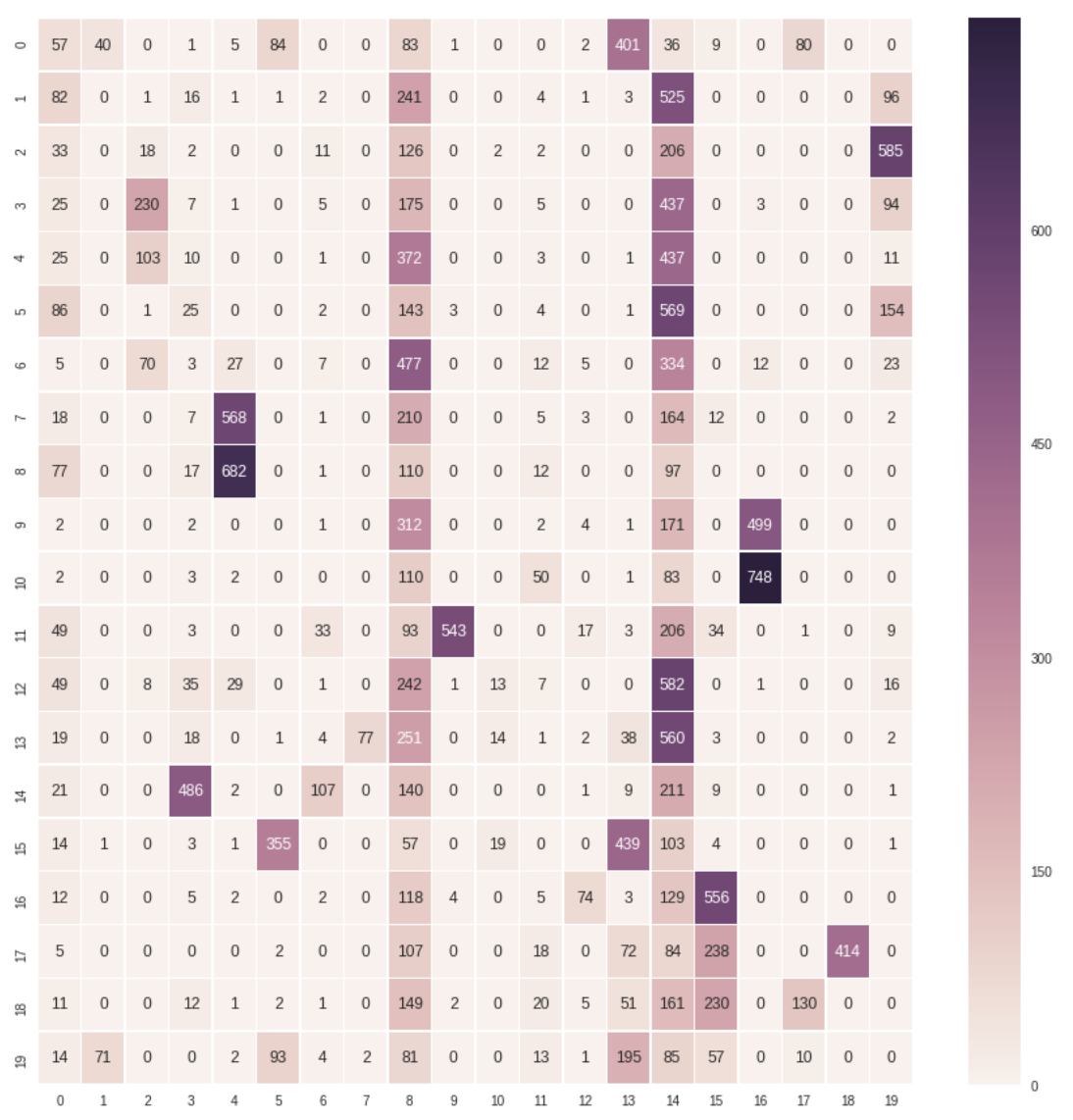
When normalization is done before logarithmic transformation the diversity of the data is lost and log transform does not help much. Even though SVD does better in this case, it should be disregarded.

Normalizing then logarithmically transforming the data causes both SVD and NMF to perform worse than if the logarithmic performance is done before the normalizing. This makes sense because the point of applying logarithmic transformation is to “un-skew” the data. And this would be most effective if applied on the entire range of data before any normalization is done.

QUESTION 11: Repeat the following for 20 categories using the same parameters as in 2-class case: • Transform corpus to TF-IDF matrix. • Directly perform K-means and report the 5 measures and the contingency matrix;

In this part we apply all the steps which we tried above to the complete the dataset of the news group articles. This dataset has 18846 documents with 52,295 terms classified into 20 categories, which we shall try to cluster.

We perform K-means clustering with K=20 over the entire data and create the contingency matrix. The Contingency matrix is as follows:



We get the following result after performing K-means:

homogeneity: 0.35942082651801804

completeness: 0.45111242050273204

v\_measure : 0.4000803165708632

adjusted\_rand\_score: 0.13663613501490818

adjusted\_mutual\_info\_score: 0.35731878968094594

QUESTION 12: Try different dimensions for both truncated SVD and NMF dimensionality reduction techniques and the different transformations of the obtained feature vectors as outlined in above parts. Report your best combination, and quantitatively report how much better it is compared to other combinations. You should also include typical combinations showing what choices are desirable (or undesirable).

On analysis we see that the direct clustering does not perform well enough. To improve the results, we try various combinations of dimensionality reduction techniques, with various transformations to search for a good clustering model.

As explained above we use the V-Measure, to measure which combination is performing better than the others. We first found the best value for r (no. of dimensions after reduction) for SVD and NMF. Using the reduced data we performed K-means to find 20 optimal clusters using different combinations of transformation and the reduction techniques.

The result of clustering improves significantly upon applying reduction and transformation. The best clustering is given when we reduce the data using truncatedSVD to 300 dimensions and transform the data using log transformation and then scale it with unit variance and finally perform K-mean clustering on it.

The metric of the best combination is:

**Transformed Then Normalized Svd Results**

homogeneity: 0.3911716351707267

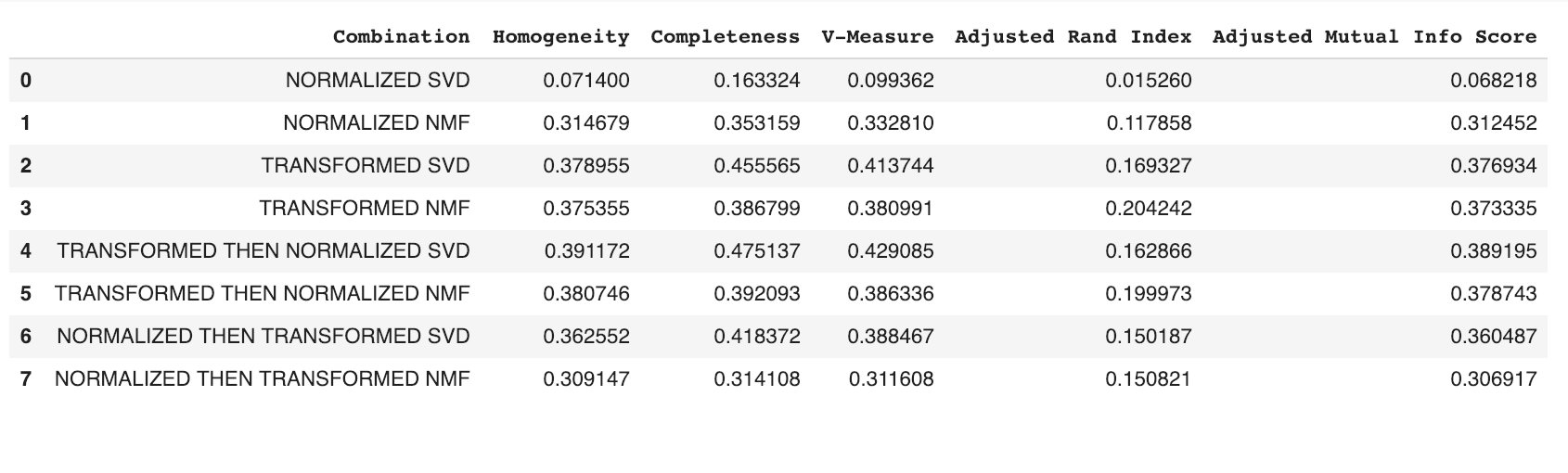
completeness: 0.4751370069430752

v\_measure : 0.4290852264443468

adjusted\_rand\_score: 0.16286551499094293

adjusted\_mutual\_info\_score: 0.3891951744095512

The other combinations that we tried to obtain desirable results are computed below in a table as follows:



Thus we see that the transformed then normalized SVD combination does the best among the other combinations.

References:

[1] <https://www.r-statistics.com/2013/05/log-transformations-for-skewed-and-wide-distributions-from-practical-data-science-with-r/>

[2]<https://www.quora.com/What-is-the-difference-between-non-negative-matrix-factorization-and-singular-value-decomposition>

[3][www.wikipedia.com](http://www.wikipedia.com) (for definitions)