

Project Report

Clustering Analysis on Textual and Image Data

Akshay Joshi
Ege Cetintas
Rohan Dutta
Shihan Gao

INTRODUCTION

Clustering is an unsupervised algorithm that is used to group together data points based on the similarity in the feature domain. Clustering is considered one of the best algorithms in the field of unsupervised learning as it deals with finding structured patterns in a collection of unlabeled data. To put it into more simpler terms, clustering is the process of grouping data into different clusters depending on how similar the data points are with respect to each other. The main difference between clustering and classification lies in the fact that classification is a supervised algorithm and clustering is an unsupervised algorithm, i.e., classification works on labeled data whereas clustering works on unlabeled data.

There are many types of clustering algorithms like K-means, Fuzzy C-Means, Hierarchical clustering, etc. We used the K-means algorithm in this project to present results.

K-means clustering

The main aim of this algorithm is to find groups in the unlabeled data depending on the features of the data points with the number of groups denoted by the variable 'K'. Each data point is associated with 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. This algorithm assigns 1 centroid to each of the k groups has a centroid to have a total of k centroids. The algorithm starts with initial random initialization of estimates of the k centroid. The algorithm then iterates between the following two steps until convergence:

1) Data Assignment Step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if c_i is the collection of centroids in set C , then each data point x is assigned to a cluster based on the distance calculated between the centroid and that particular data point. It can be written as follows:

$$\text{Minimize } J = \sum_{n=1}^N \sum_{i=1}^K ||x_n - c_i||^2$$

The distance is basically the L2 or the Euclidean distance between the n th datapoint (x_n) and the centroid it is associated with i.e. c_i .

2) Centroid Update Step:

In this step, the centroids' positions are recomputed. The centre of the cluster, i.e, the centroid's position is set to the mean of the data points that are currently within the cluster.

The k-means algorithm iterates between these two steps until a stopping criteria is met (when no data points change clusters, maximum number of iterations reached, sum of distances is minimized).

DATASET

The dataset that we used for the execution of this project is the '20 Newsgroup' dataset. It is a collection of approximately 20000 newsgroup documents, partitioned evenly across 20 different newsgroup categories. For this project's purpose, we used documents categorized into 8 categories viz. comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.graphics, rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey. These categories are selected since they are closely related to each other. Following table shows the next division of these categories into two different categories.

Computer Technology	Recreational Activity
comp.os.ms-windows.misc	rec.autos
comp.sys.ibm.pc.hardware	rec.motorcycles
comp.sys.mac.hardware	rec.sport.baseball
comp.graphics	rec.sport.hockey

In order to define this problem for clustering analysis, we pretended the dataset to be unsupervised, meaning, the class labels to be absent. Then we performed clustering analysis 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 used the labels as the ground truth to evaluate the performance of our clustering model.

FEATURE EXTRACTION

This is an important step in any type of classification. Feature extraction entails extracting important information from the data which proves to be crucial in classification. In text classification particularly, we need to have a representation of documents. We used a 'Bag of Words' model for the purpose. It is a simplifying representation used in natural language processing and information retrieval. In this model, a document is represented by the frequency of occurrence of each word. We used Term Frequency-Inverse Document Frequency commonly known as TF-IDF as a metric to evaluate how important a word is to a document in a collection or corpus. This relates to the importance of a word being directly proportional to the number of times it appears in the document but is offset by the number of times the word appears in the entire corpus. For example, if the corpus is all about soccer, then words like 'shoes', 'football', 'kick', etc. will be present in almost every document and thus these kinds of words would not be distinguishing factors in classification. TF-IDF is composed of two terms. TF and IDF. TF of term 't' is calculated by $TF(t) = (\text{Number of times term 't' appears in a document}) / (\text{Total number of terms in the document})$. While computing TF, all terms are considered equally important. However it is known that certain terms, such as 'is', 'of', and 'that', may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the IDF as $IDF(t) = \log(\text{Total number of documents} / \text{Number of documents with term t in it})$.

Tokens are basically individual units of meaning that we are operating on. We convert the textual data into these tokens by a process called tokenization. Tokenization is the process of breaking down text documents apart into those pieces. In text analytics, tokens are most frequently just words. We implemented preprocessing of data by converting the collection of text documents to a matrix of token counts using sklearn's **CountVectorizer** module. We passed the tokenizer parameter as our lemmatized set of words with numbers and digits removed. We also removed English stop words which prove to be of little importance due to the high frequency of occurrences in the documents. For example, 'a', 'an', 'and', 'there', etc. Also, we set the minimum document frequency as 3, i.e. the words with a minimum frequency of 3 were only considered as features. After preprocessing both the training data and the testing data by passing them to the CountVectorizer

module, we obtained the count matrix of both the training dataset and the test dataset. We passed the count matrix into the **TfidfTransformer** module and performed feature extraction. The module outputted the TF-IDF matrices of the datasets which are as shown in the figure below:

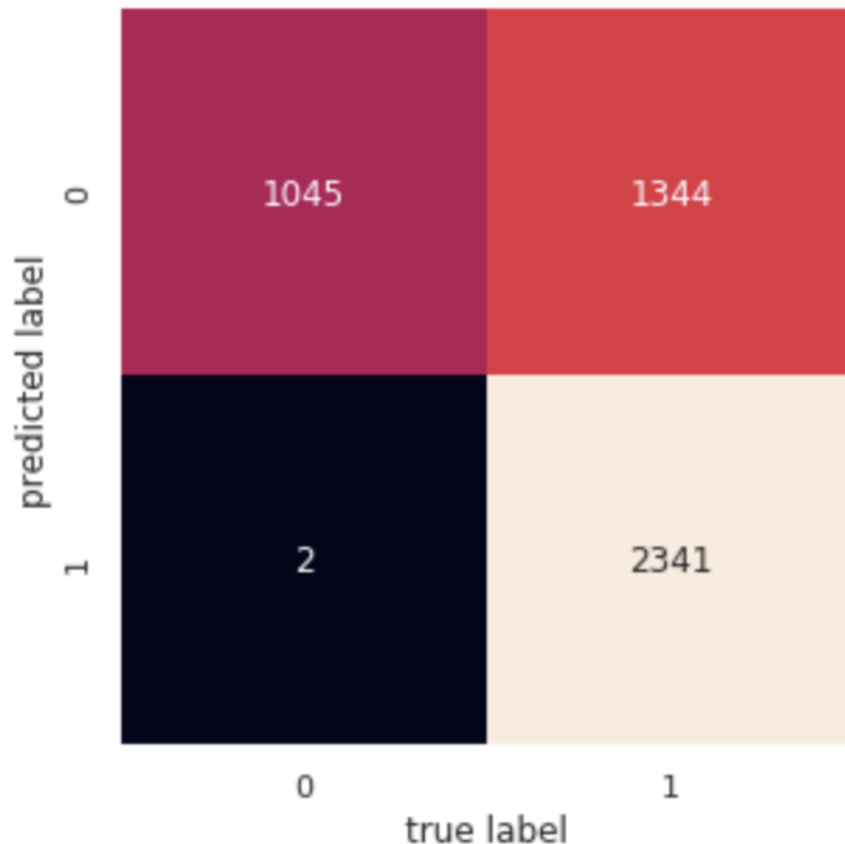
```
[11] print("Shape of TRAIN TF-IDF matrix: ",X_train_tfidf.shape)
      print("Shape of TEST TF-IDF matrix: ",X_test_tfidf.shape)
```

```
↳ Shape of TRAIN TF-IDF matrix: (4732, 16432)
   Shape of TEST TF-IDF matrix: (3150, 16432)
```

IMPLEMENTATION OF THE ALGORITHM

We implemented the k-means clustering algorithm using sklearn's KMeans class. Since we have created a data table having only two classes, we aim to cluster the documents into two classes. Therefore, in this case, the value of k is equal to 2.

After implementing the algorithm, we tested the results obtained using the ground truth values. We obtained the contingency matrix which is the frequency distribution of the data points with respect to the actual label value and the clustered result. Following is the table that we obtained:



A heatmap representing a contingency matrix. The vertical axis is labeled 'predicted label' with values 0 and 1. The horizontal axis is labeled 'true label' with values 0 and 1. The cells contain the following counts: (0,0) is 1045 (dark red), (0,1) is 1344 (red), (1,0) is 2 (dark blue), and (1,1) is 2341 (light orange).

	0	1
0	1045	1344
1	2	2341

As it can be seen from the contingency matrix, it is visible that there are 3386 documents that have been correctly clustered and the remaining 1346 documents are not clustered correctly. The contingency matrix is just one way to evaluate the clustering task's performance. There are various other metrics that are used to

evaluate the model's performance. Some of them are homogeneity, completeness, V-measure, adjusted rand index, and adjusted mutual information score.

Homogeneity is a measure of how pure a cluster is. In other words, if each cluster contains data points from a single class, then homogeneity is satisfied.

Completeness is satisfied if all data points of a class are assigned to the same cluster.

The V-measure is defined as the harmonic mean of homogeneity and completeness.

Rand index is a measure of the similarity between two data clusterings.

The adjusted mutual information score measures the mutual information between the cluster label distribution and the ground truth label distributions.

THE 5 MEASURES

Following snapshot tells the score values of each of the 5 parameters/metrics.

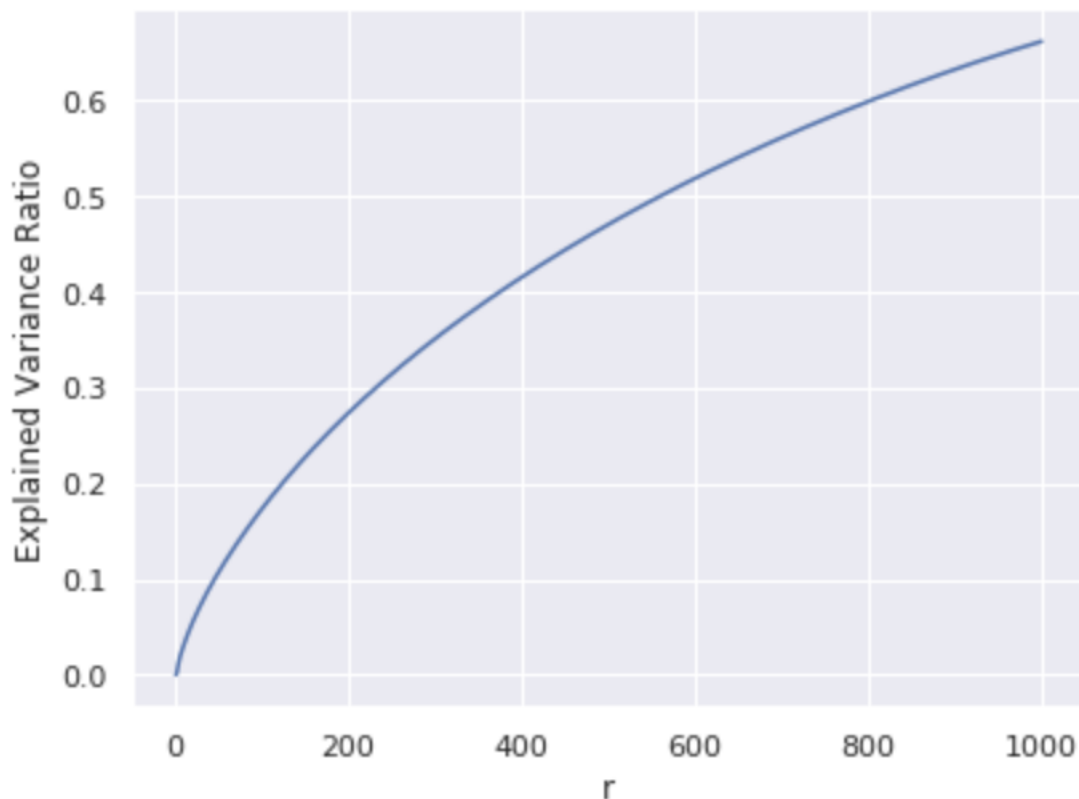
```
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels, km.labels_))
print("Completeness: %0.3f" % metrics.completeness_score(labels, km.labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(labels, km.labels_))
print("Adjusted Rand-Index: %.3f"
      % metrics.adjusted_rand_score(labels, km.labels_))
print("Adjusted Mutual Info Score: %0.3f"
      % metrics.adjusted_mutual_info_score(labels, km.labels_))
```

```
Homogeneity: 0.258
Completeness: 0.339
V-measure: 0.293
Adjusted Rand-Index: 0.186
Adjusted Mutual Info Score: 0.293
```

DIMENSIONALITY REDUCTION

The clustering result was not optimum as is evident by the performance metrics. Higher dimensional TF-IDF data does not yield good clustering results. We use dimensionality reduction techniques to find top singular values which correspond to the most significant features (which contribute by a large margin to the data).

The dimensions of the output TF-IDF matrices that we got after performing TF-IDF transformation were of a much higher order, thousands to be precise as evident by the screenshot. It is known that higher number of dimensions, or in other words, more number of features lead to reduction in the model performance. This phenomenon is known as the curse of dimensionality. The more dimensions you add to a data set, the more difficult it becomes to predict certain quantities. Therefore it is necessary to reduce the dimension of the dataset to improve the performance and prediction value. In this project we used two methods to reduce the dimensionality viz. Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF). We plotted the graph of the first r principal components against the variance that it could retain. We varied the value of r from 1 to 1000. The plot was obtained as follows.



Reinstating the fact that the value of r is the number of first r principal components. As can be seen from the plot, as r increases, the variance ratio increases (kind of) logarithmically. Therefore, we can say that increased value of r will lead to higher variance ratio. But as we move further down the analysis, it can be seen that with increased value of r , the performance of the model decreases. Thus, there is a tradeoff between increasing the value of r which means increasing the level of information preservation, and the performance of the model. We performed dimensionality reduction using Truncated SVD/Principal Component Analysis (PCA) and Non-Negative Matrix Factorization (NMF).

PRINCIPAL COMPONENT ANALYSIS

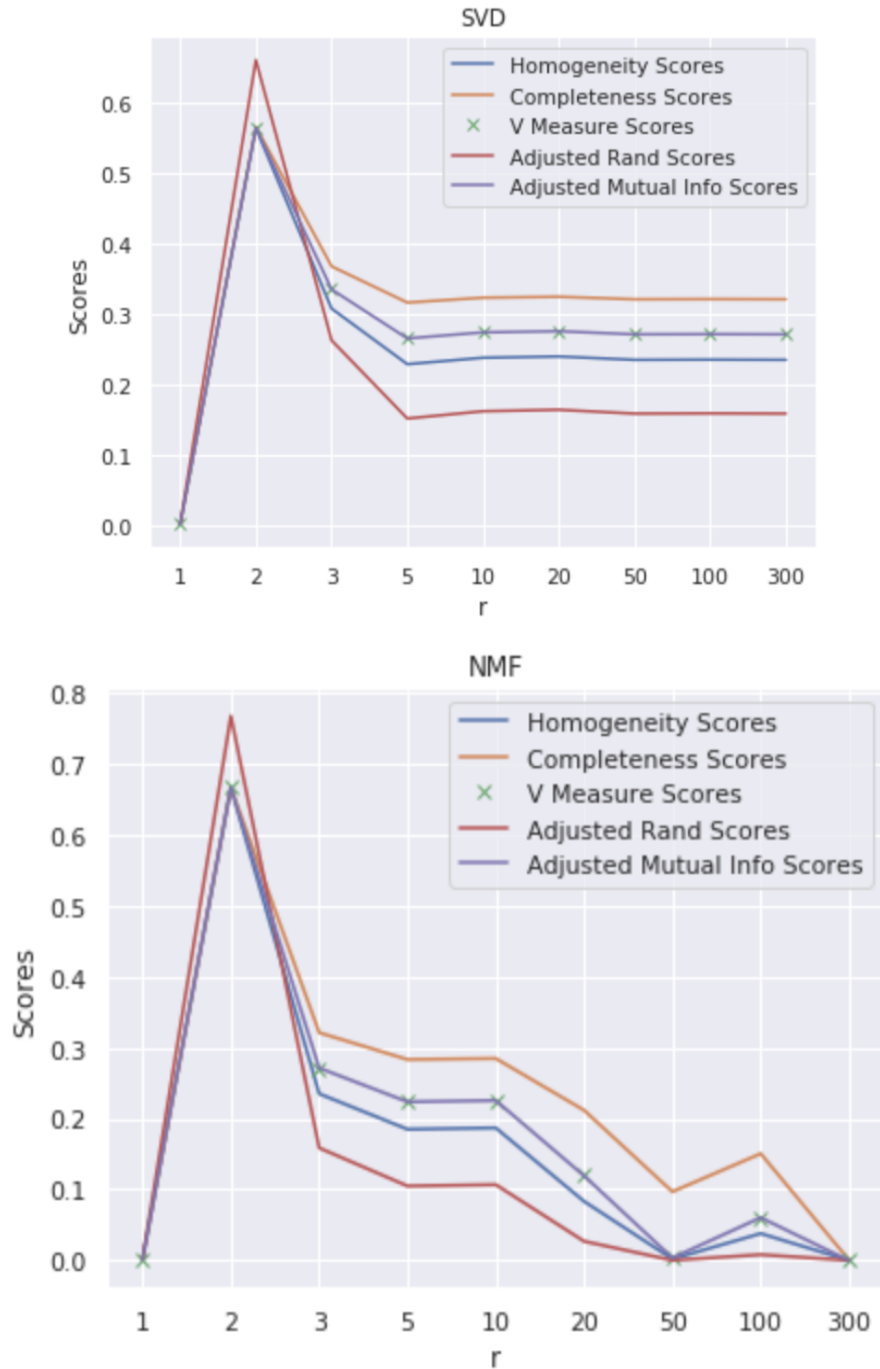
Principal Component Analysis (PCA) is essentially a coordinate transformation. The original data are plotted on an X -axis and a Y -axis. For two-dimensional data, PCA seeks to rotate these two axes so that the new axis X' lies along the direction of maximum variation in the data. PCA requires that the axes be perpendicular, so in two dimensions the choice of X' will determine Y' . The transformed is obtained data by reading the x and y values off this new set of axes, X' and Y' . For more than two dimensions, the first axis is in the direction of most variation; the second, in direction of the next-most variation; and so on. This new set of axes are basically formed using the eigenvectors of the original data matrix.

NMF

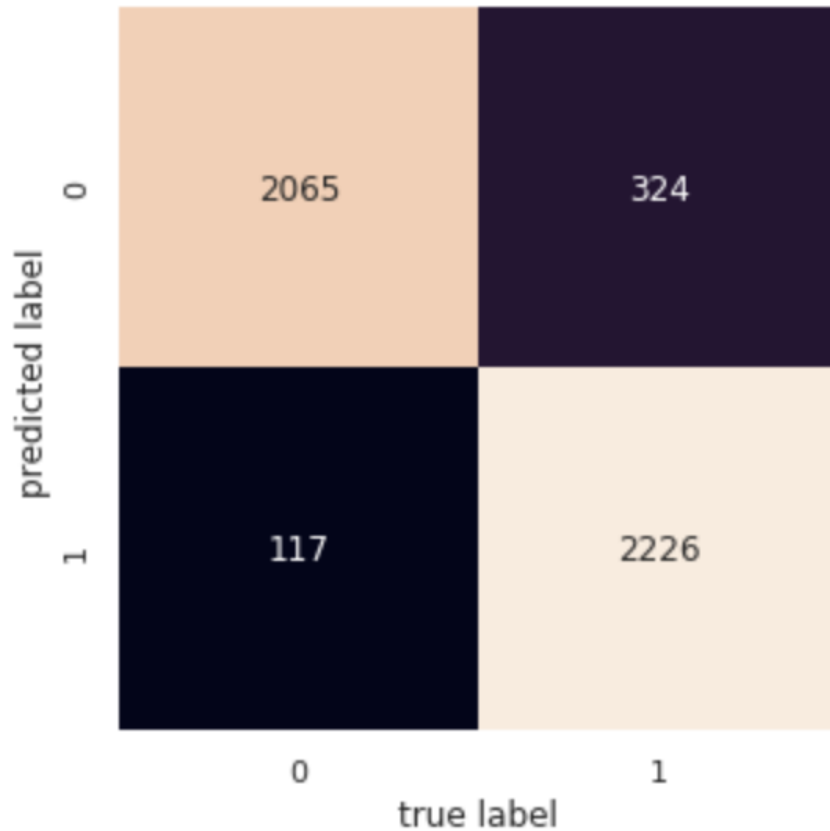
Non-Negative Matrix Factorization can also be used to reduce the dimensionality. The matrix X is decomposed to two non-negative matrices W and H such that $\|X - WH\|_F^2$ is minimized. Then we used W as the dimension reduced data and in the fit step we calculated both W and H . We performed this factorization to break down X 's rows which are the number of documents into a linear combination of r topics.

We then set the value of r to a set of fixed numbers and calculated the 5 measure scores for those values of r for PCA as well as NMF. More specifically, we calculated the 5 measures for $r = 1, 2, 3, 5, 10, 20, 50, 100, 300$ and plotted the graphs for both the dimensionality reduction techniques.

In these following graphs it can be clearly seen that as we increase the value of r , i.e., as we increase the level of information preservation, it doesn't mean that the performance increases. On the contrary, as we increase the level of information, the performance of the model decreases. Following are the plots of the different values against the 5 measure scores for both the techniques.



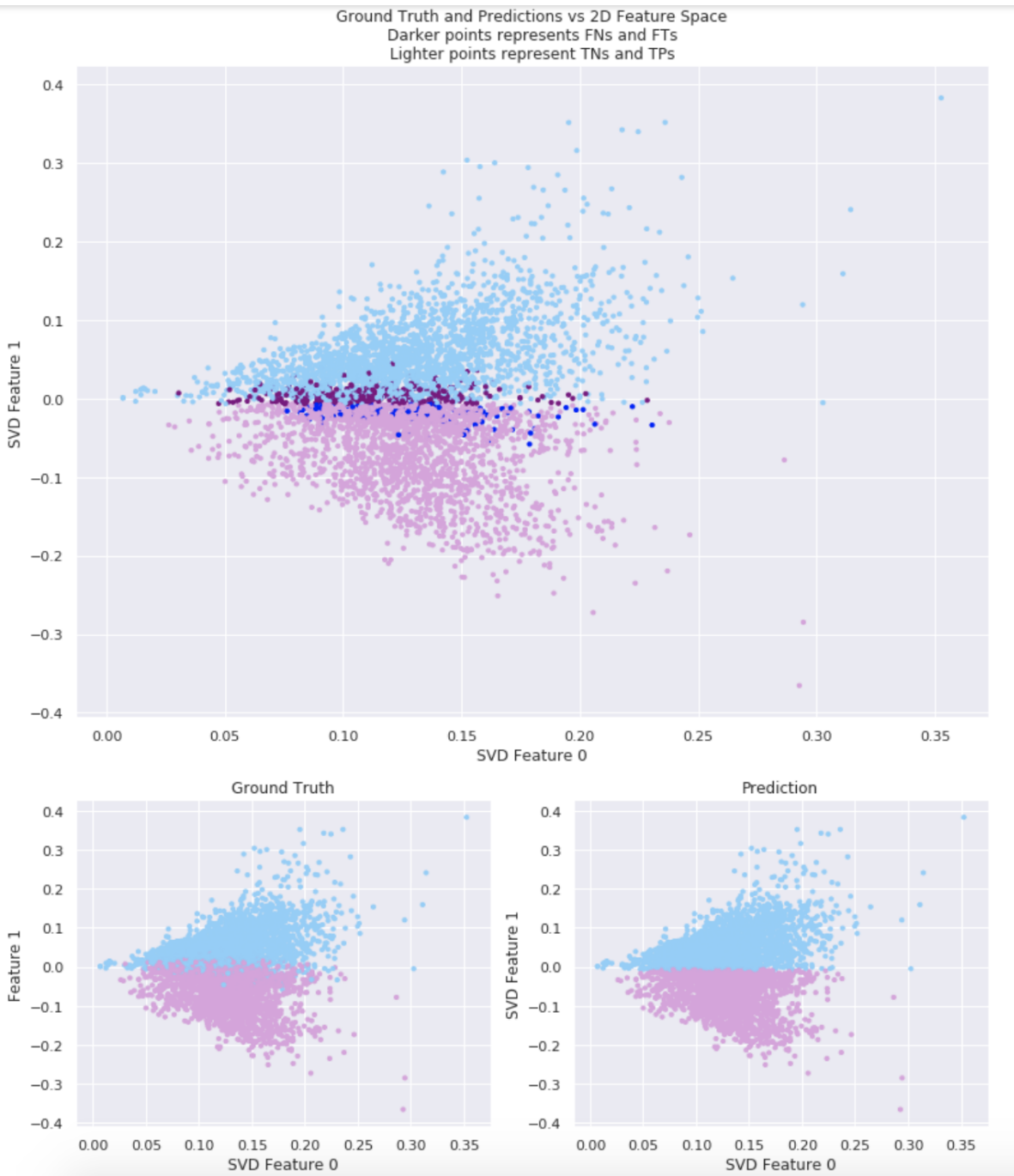
Therefore, after analyzing the graphs, it can be seen that $r = 2$ is the best choice since it outputs the maximum scores. Using $r = 2$, we plotted the contingency matrices for both the techniques which happened to be the same.



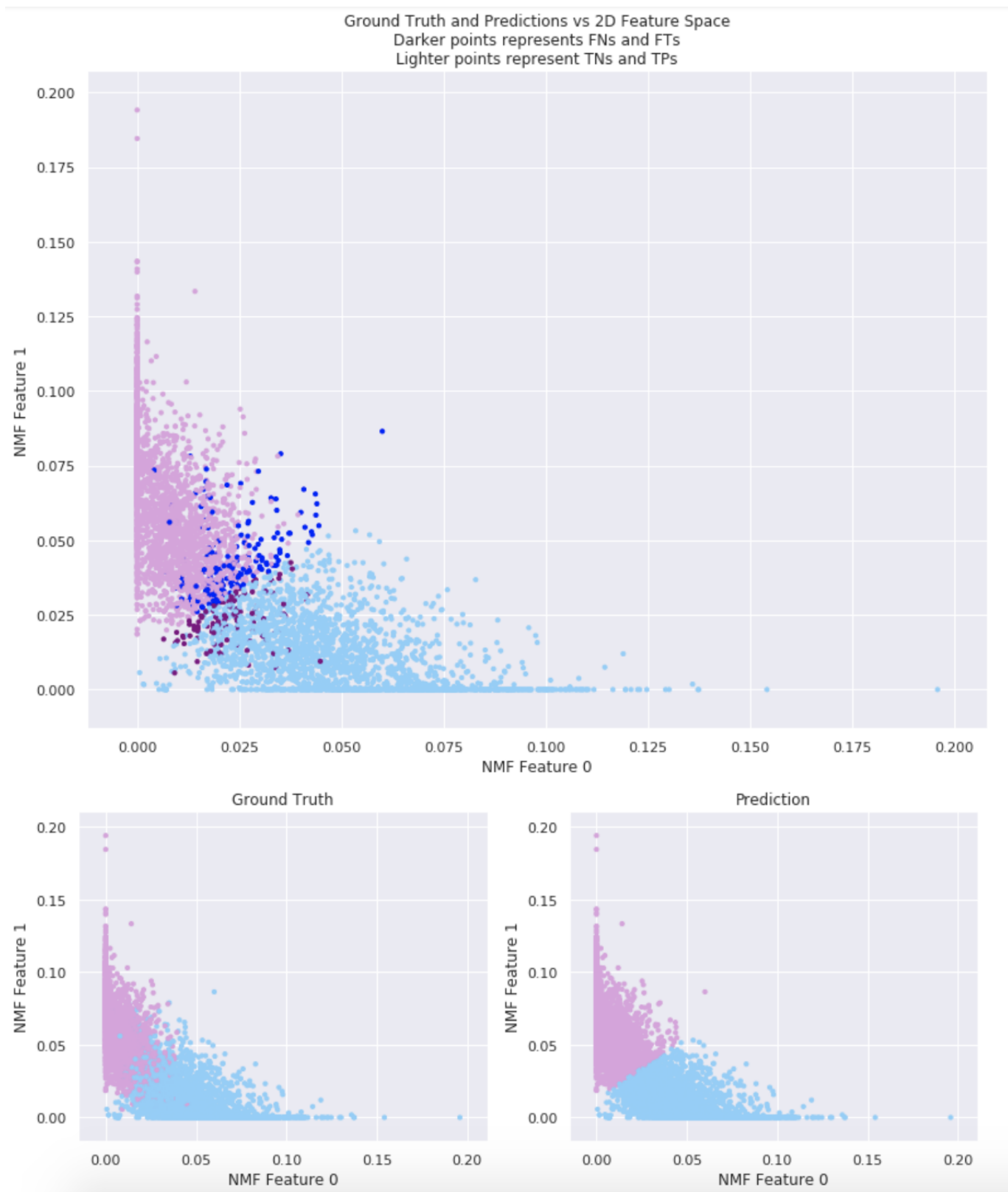
There was a significant improvement in the number of wrongly clustered documents. Without dimensionality reduction, the model wrongly clustered 1346 documents. With dimensionality reduction, the number reduced to 441. Therefore using $r = 2$, we significantly increased the model performance.

Increase in value of r is the same as increase in the number of components. This means there are more elements to cluster. Therefore, when we increase the value of r beyond the optimal value, the distance metrics that are used to cluster data points become kind of redundant. Therefore, as r increases, the function becomes non-monotonic.

We visualized the clustering results for both SVD and NMF by plotting various figures depicting the ground truth, predicted result and both of them combined for better analysis of our results.



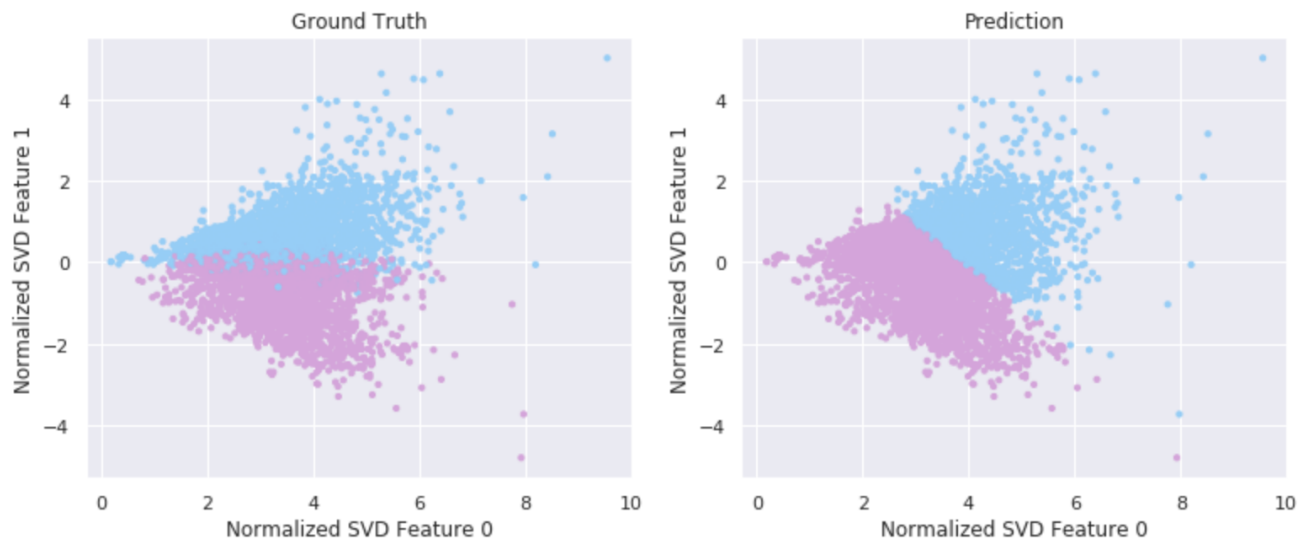
VISUALIZATION RESULTS FOR SVD ($r = 2$)



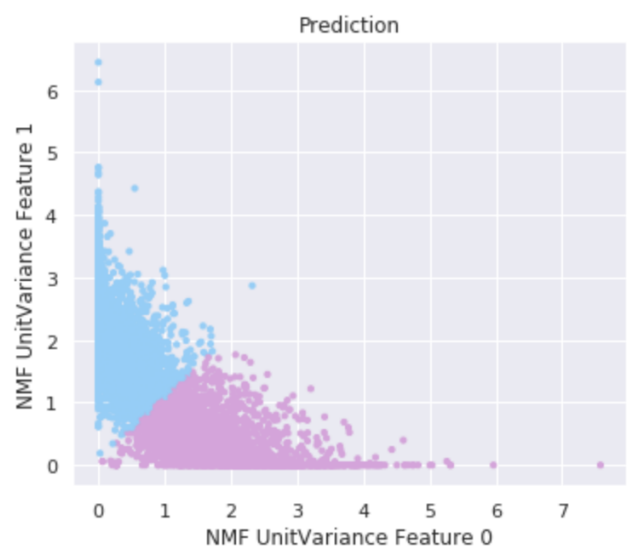
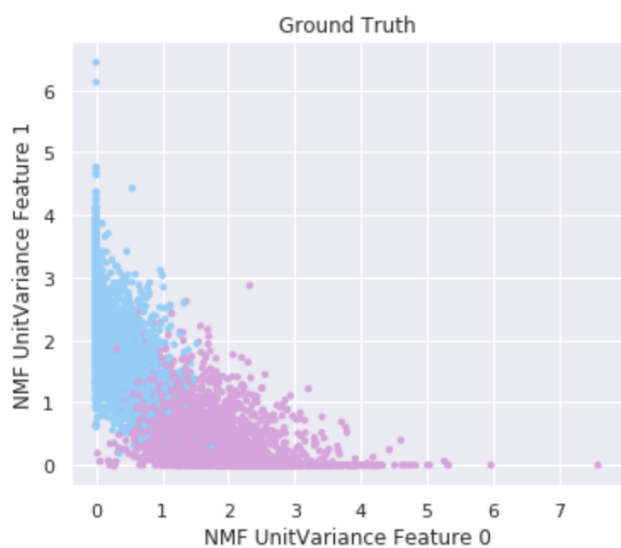
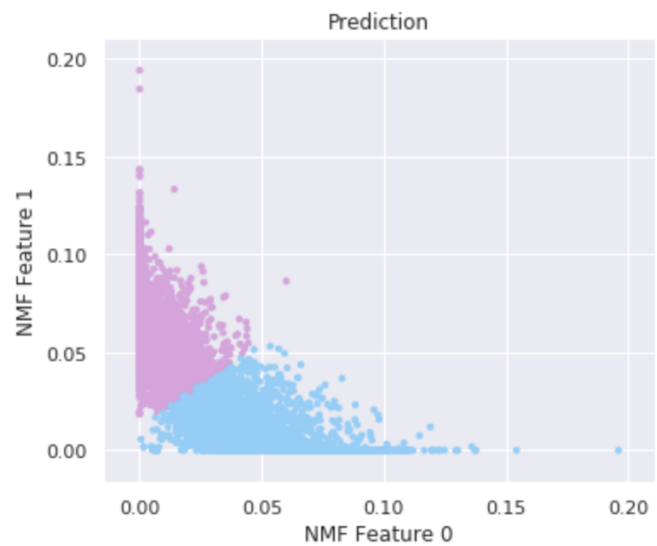
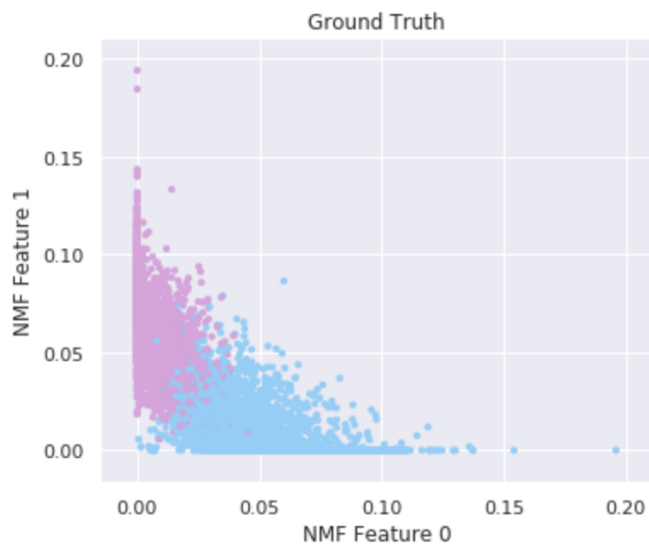
VISUALIZATION RESULTS FOR NMF ($r = 2$)

The light blue and the plum coloured data points represent the data points that have been correctly clustered. The darker ones, i.e. purple and dark blue represent the data points that have been incorrectly clustered. The two plots below the main plot show the ground truth and the prediction. The lighter colours represent true positives and true negatives and the darker ones represent the false positives and false negatives. The plot makes sense because the data points nearer to the boundary of the cluster have a higher chance of being wrongly clustered.

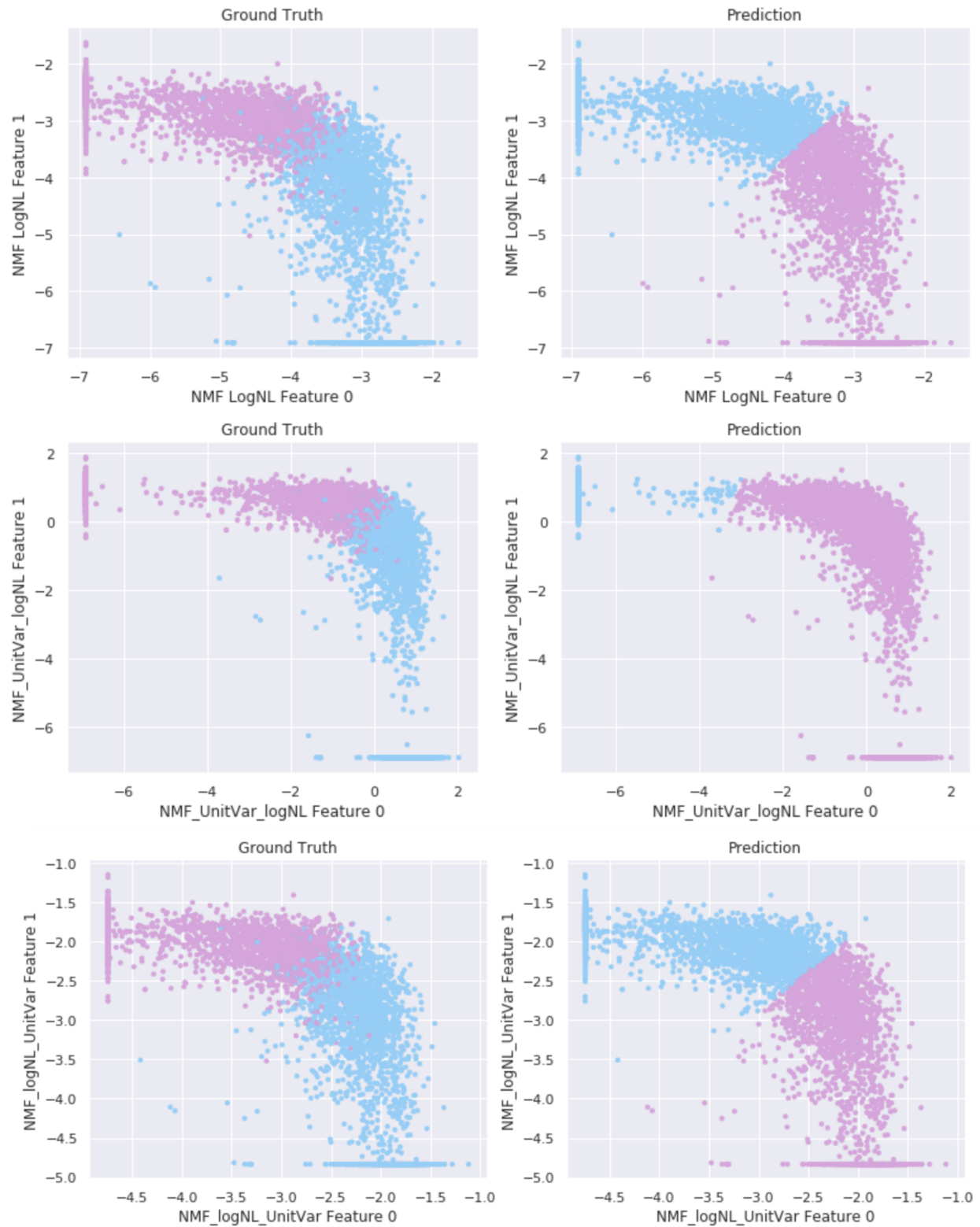
Further, we performed transformation methods like feature scaling and applying logarithmic non-linear transformation to our reduced data matrix. We scaled features such that each one would have a unit variance. We obtained the following results.



FEATURE SCALING FOR SVD

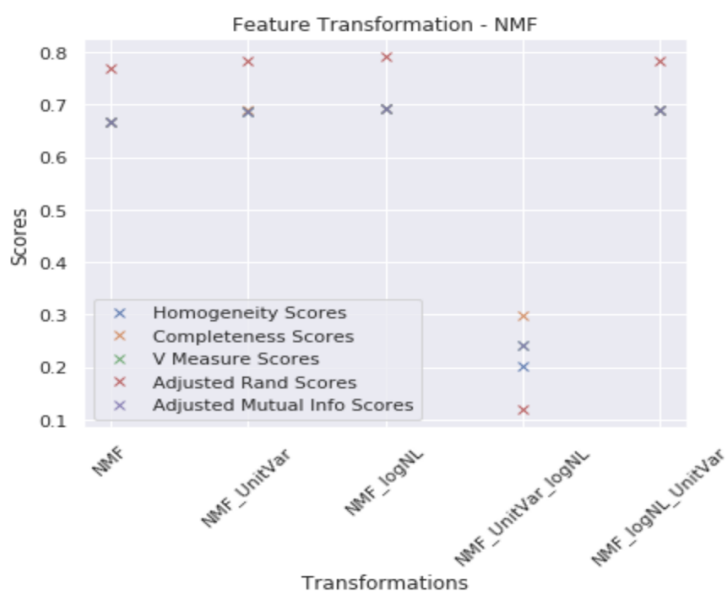
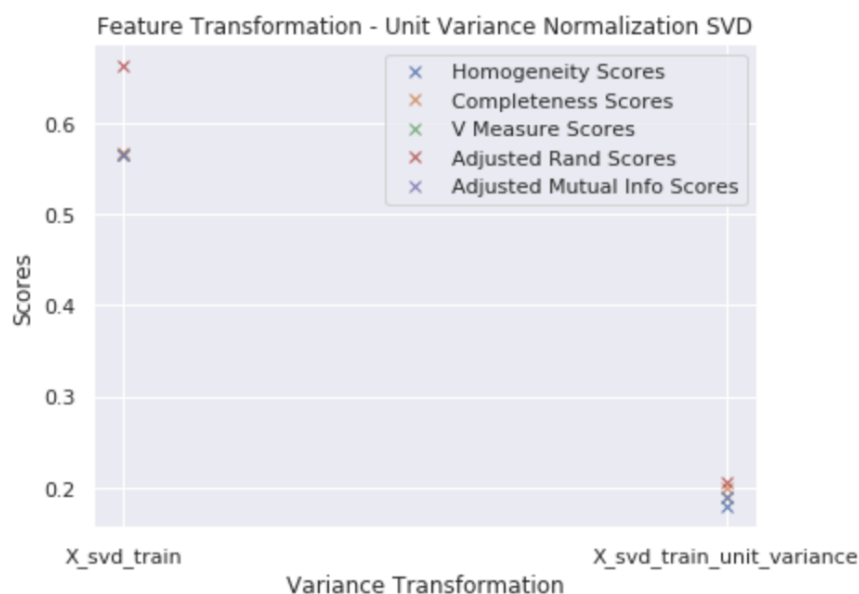


FEATURE SCALING FOR NMF



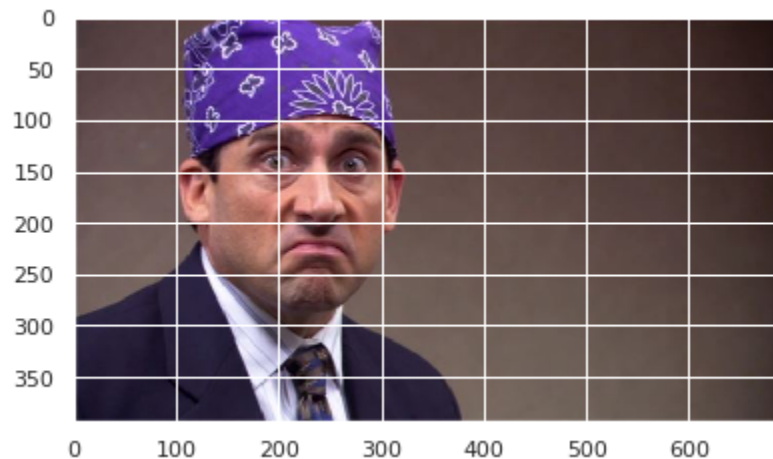
FEATURE SCALING AND LOGARITHMIC TRANSFORMATION FOR NMF

Different scale of data could affect the distance between the data points. Logarithm transformation makes data more spread out or less skewed. Good use of k-means clustering assumes the variables with more symmetric distributions and the algorithm often shows poor performance unless data groups have spherical shapes and approximately same sizes. As the scale is changed to logarithm, the relative distance between the points tends to be larger, the data points tend to be distributed more symmetrically and more sparsely thus the probability of incorrect clustering of data points near the boundary is decreased.



COLOR CLUSTERING

Image segmentation is an intermediate image processing stage in which the pixels of the image are grouped into clusters. Image is reshaped to a matrix of size $394 \times 697 \times 3$, where 394×697 denotes the number of pixels and the size 3 represents the RGB channels. The original image is shown below and here we use functionalities in Matplotlib package for loading, rescaling and displaying images.

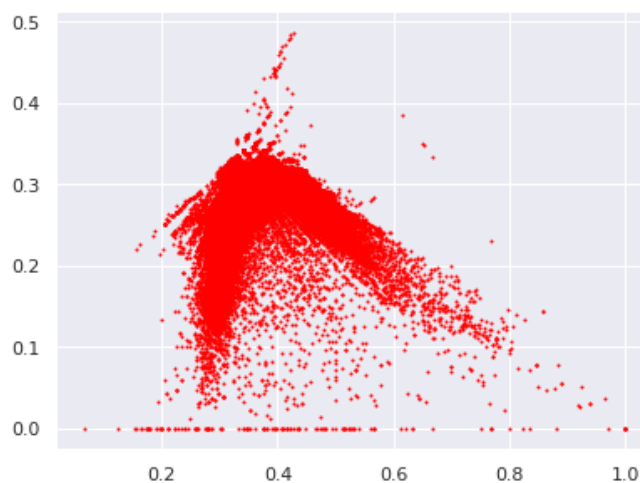


The pixel RGB information is first transformed to “normalized (r, g) space”.

```
[ ] def normalizeImage (img):  
    #return img/np.dstack((np.sum(img,axis=2),np.sum(img,axis=2),np.ones(img[:, :, 2].shape)))  
    sumRGB = np.sum(img,axis=2)  
    sumRGB[sumRGB == 0] = 1  
    return img/np.dstack((sumRGB,sumRGB,sumRGB))
```

Here is the scatter plot of imgG (G channel) vs imgR (R channel) in the normalized (r,g) space, the size of which is both 394×697 .

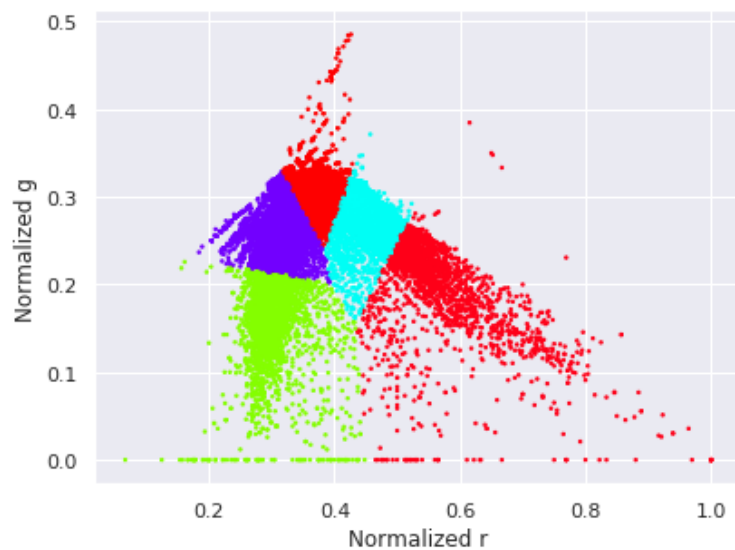
$$r = \frac{R}{R+G+B}, g = \frac{G}{R+G+B}.$$



The number of clusters is chosen to be 5, and colors of pixels are clustered. Here we still use the k-means clustering. Best cluster centers are those that minimize sum of square distance between all points and their nearest cluster center. K-means clustering based on intensity or color is essentially vector quantization of the image attributes.

```
numberClusters = 5
km = KMeans(n_clusters=numberClusters, max_iter=1000, n_init=30, verbose=False, random_state=0)
X = np.concatenate((imgR.reshape(imgR.shape[0],1),imgG.reshape(imgG.shape[0],1)),axis=1)
km.fit(X)
```

Here we compute the k-means clustering and plot the scatter of clustered imgR vs imgG and each color denotes each cluster.



Then we scatter plot imgY vs imgX which represent the pixels of the image, and each pixel with labels of the clustered color information.

