Project 2
Clustering

Donna Branchevsky
UID: 404473772

Pavan Holur
UID: 204403134

Megan Williams
UID: 104478182

Tadi Ravi Teja Reddy
UID: 505227246

University of California, Los Angeles

Winter 2019

Question 1

Importing the news groups from the scikit-learn library into python's working directory, we observed the 20 classes of data. The classes were labelled by context, with the data modelled as text snippets associated with each label.

The categories extracted from the data set for binary classification were sourced as follows:

1. Among the 20 classes, 8 classes were selected and grouped into two broader classes. The eight sub classes were: 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey'.

2. The first 4 and last 4 classes unionized to broader classes named 'Computer Technology' and 'Recreational Activity' respectively.

This process resulted in 2 large class labels comprising of 4 base labels each. Note from Question 1 that the two super-labels formed had balanced data sets. This process was accomplished by using the *fetch_20newsgroups* function again, with specified categories from above. Once obtained, the features used for each data point was a "Bag of Words" model, which counts the frequency of words in a sample data point.

Once the large feature vectors were formed, we ignored features that were passed by the *CountVectorizer*, but still were not distinguishing factors between the two binary classes, because of their contextual dominance in overpowering our classification. Note that the min_df setting in the *CountVectorizer* ignored the noise in the text data by avoiding bags for which the word count was < 3. To avoid the bias, we further constructed the TF-IDF vectors: normalized vectors that removed implicit offset in the data. The bag-of-words features from the *CountVectorizer* were now normalized through the TF-IDF transform, yielding feature vectors. The resulting data matrix dimensions were: (the target variable is simply a 1D vector with the binary target values).

- X_tfidf size: (7882, 27768)

Question 2

We then applied K-Means clustering with $k = 2$ using the TF-IDF data,. The settings were set as follows: (Note: the init_clusters have been set to KMeans++, which is an optimized method used by the library to initialize clusters yielding on average faster convergence).

```
kmeans = KMeans(n_clusters=2,  n_init=30, max_iter=1000,  random_state
    =0).fit(X_tfidf)
pred = kmeans.predict(X_tfidf)
```

In this case we know the number of clusters because we are aware of the training data labels. This is counter-intuitive to the primary purpose of clustering, which is fundamentally unsupervised learning.

The resulting contingency matrix is expected to be weak; there are several issues with raw clustering with K-Means, including scaling, offset, isotropic nature of clusters etc.. The contingency matrix is given below:

	Cluster A	Cluster B
Class 1	4	3899
Class 2	1718	2261

Table 1: Performance of K-Means clustering algorithm

The contingency matrix indicates that while cluster A is reasonably controlled in letting only class 2 elements, cluster B is less discerning; there are many elements of both classes in the cluster (this indicates a poor classification).

Question 3

After obtaining our clustering results, we performed five different evaluation methods to determine how good they are, in covering our training data w.r.t to the labels.

- **Homogeneity** - Satisfied if each cluster contains only data points from a single class.
- **Completeness** - Satisfied when all data points of a class are assigned to the same cluster.
- **V-measure** - The harmonic average of homogeneity score and completeness score."
- **The adjusted Rand Index** - Sum of all pairs of points that both fall either in the same cluster and the same class or in different clusters and different classes.
- **The adjusted Mutual Information Score** - Measure of the mutual information between the cluster label distribution and the ground truth label distributions.

To perform these evaluation methods, we used functions provided by sklearn.metrics:

```python
def print_metrics_and_return(Bin_Target, pred):
    m1 = (homogeneity_score(Bin_Target, pred))
    m2 = (v_measure_score(Bin_Target, pred))
    m3 = (completeness_score(Bin_Target, pred))
    m4 = (adjusted_rand_score(Bin_Target, pred))
    m5 = (adjusted_mutual_info_score(Bin_Target, pred))
    return list([m1, m2, m3, m4, m5])

print(print_metrics_and_return(Bin_Target, pred))
```

We obtained the following results:

Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
0.25359589	0.28860034	0.33481575	0.18076180	0.25352755

Table 2: Performance of K-Means clustering algorithm

As predicted, our model is poor. The comparison will become more evident in proceeding sections when we observe improvement in these metrics.

Question 4

From our evaluation in question 3, we observed that our high dimensional, sparse, TF-IDF vectors do not yield good clustering results. This can be partially attributed to the Euclidean distance metric not performing well with high dimensional data. Because of this, we use dimensionality reduction techniques, Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF), before clustering to get data that better suits the K-Means clustering algorithm.

```
tsvd = TruncatedSVD(n_components=1000, n_iter= 30, random_state=42).fit(
    X_tfidf)
X_shrunk = tsvd.transform(X_tfidf)
#print(X_shrunk)
print(tsvd.explained_variance_ratio_)
A = [(tsvd.explained_variance_ratio_[0:i]).sum() for i in range(1, 1001)
    ]
```

We applied SVD to the TF-IDF matrix. We took the top r principal components, and calculated the variance ratio of the reduced data to the original data. The variance ratio is plotted below for r from 1 to 1000.

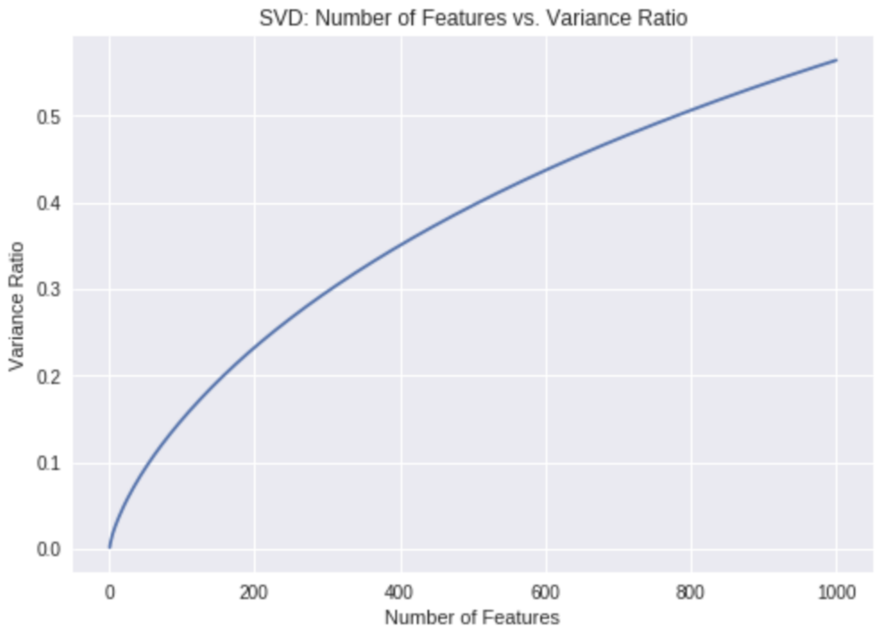
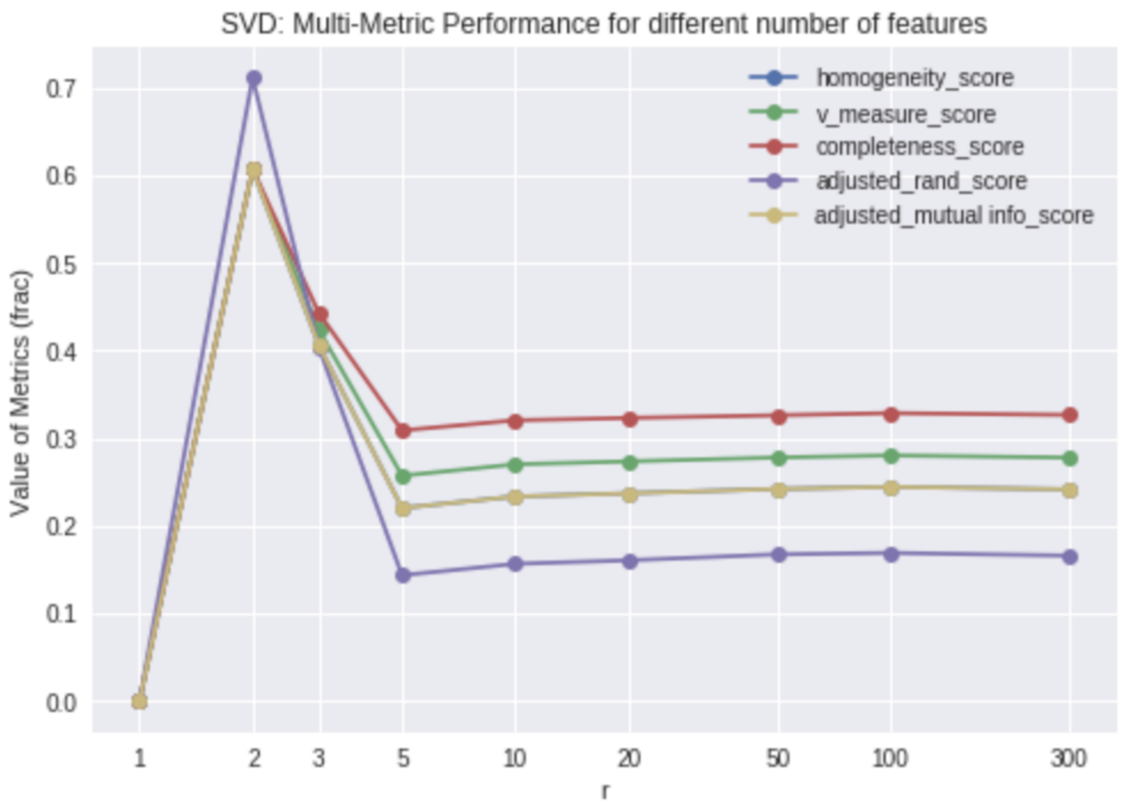


Figure 1: Cumulative Variance Ratio with respect to number of features

This graph shows that our notion about SVD is correct; the first few singular values in the decomposition capture the majority of the variance in the data. Using more features might corrupt the distances. Therefore the goal is to find the right number of features (r) to enable best K-Means clustering.

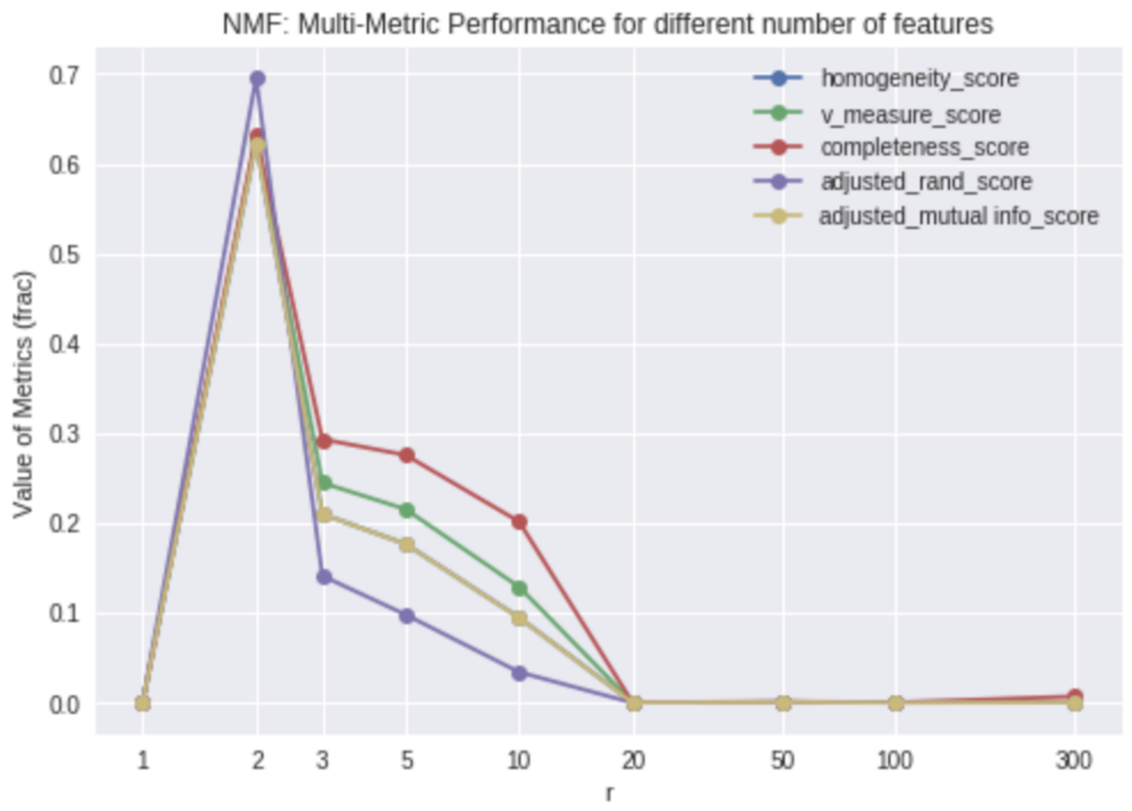
Question 5

We performed K-Means clustering on the dimension-reduced data. To find the best dimension r to reduce the data to, we ran the K-Means clustering algorithm on the data using $r = 1, 2, 3, 5, 10, 20, 50, 100, 300$. We tested the two different dimensionality reduction techniques, SVD and NMF. The performance of the K-Means algorithm is shown in the plots below for both SVD and NMF over various r. The performance is measured based on the 5 metrics listed in Question 3.



We found that using $r = 2$ for both SVD and NMF gave the best performance across all metrics. The process was validated by the fact that our selection of the range of r ensured local maximum in the performance metrics, as promoted in Question 6. Thus our range of r is not skewed.

Lastly, in this section we did not have to take into account the independent metrics; as seen in the graphs above, all measures peaked at $r = 2$. Even otherwise, a good measure for the classification would be V_measure: the average of the homogeneity and completeness score. Reasons why this is a good measure include its inherent symmetry, class invariance and the harmonic construction of 2 unique measures.



Question 6

For a smaller number of features, we under-fit to the binary clustering samples, losing out on accuracy to correctly bin predicted label to true label (i.e the implicit loss in variance results in poorer classification). For an excessive number of features, the model is over-fitted to features that DO NOT distinguish between the labels. Their contribution to the Euclidean distance measure in the K-Means algorithm can considerably offset the distances of all training points, making those features with high variance and smaller distance measure irrelevant in the clustering. This results in loss of accuracy. Thus we observed the non-monotonic behavior of measures as r increases.

Question 7

We found that best r for NMF and SVD was 2. In this question we try to visualize the K-Means cluster algorithm predictions with the actual class labels. We do this by projecting the dim-reduced data points onto 2-D plane with SVD, and coloring the points according to:

1. Clustering label
2. Class label

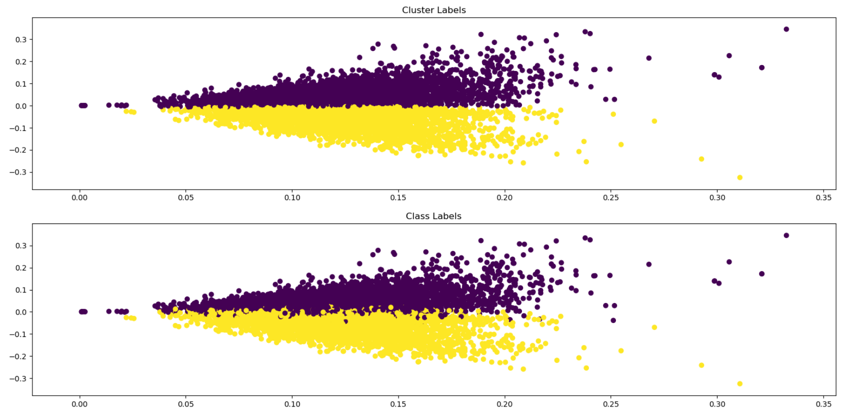


Figure 2: K-Means for SVD

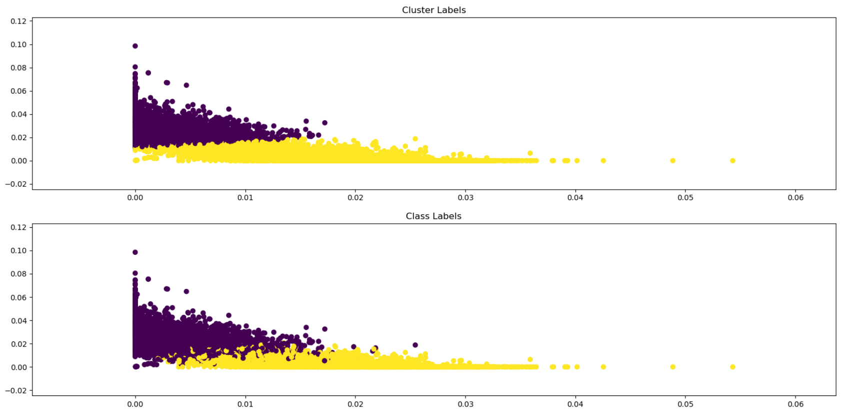


Figure 3: K-Means for NMF

Note: All the data points for NMF are positive, as expected.

Question 8

Next, we transform the data to improve the clustering performance. We implement the following two transformations in our project:

1. Scaling features so that each feature has unit standard deviation and zero mean.

2. Logarithm transformation

We define logarithm transformations in our project as follows:

$$f(x) = sign(x) \times (\ \log(\ |\ x\ | + c\ ) - \log c\ ) \tag{1}$$

```python
def scalingfeatures(X):
    X_scaled = preprocessing.scale(X, axis=0, with_mean=True, with_std=True)
    return X_scaled

def logscale(X, c):
    X_sign = np.sign(X)
    X_abs = np.absolute(X)
    return X_sign * (np.log(X_abs + c) - np.log(c))
```

There would be a total of 4 different combinations per class:

1. Mean and variance scaling

2. Logarithm transformation

3. Mean and variance scaling followed by logarithm transformation

4. Logarithm transformation followed by mean and variance scaling

We perform these operations for both, SVD and NMF. We plot all of these 8 combinations along to assess the performance of the K-Means on each of the cluster.

Note: We consider $c = 0.01$ for logarithm transformation in our project and r is 2 for both, SVD and NMF.
From the above 4 combinations on SVD and NMF, we conclude that follwing:

1. SVD with logarithm transformation and SVD with logarithm transformation followed by mean and variance scaling perform the best.

2. NMF with the all possible combinations gave equally good results, it might be helpful to look at the metrics to decide the best combination.

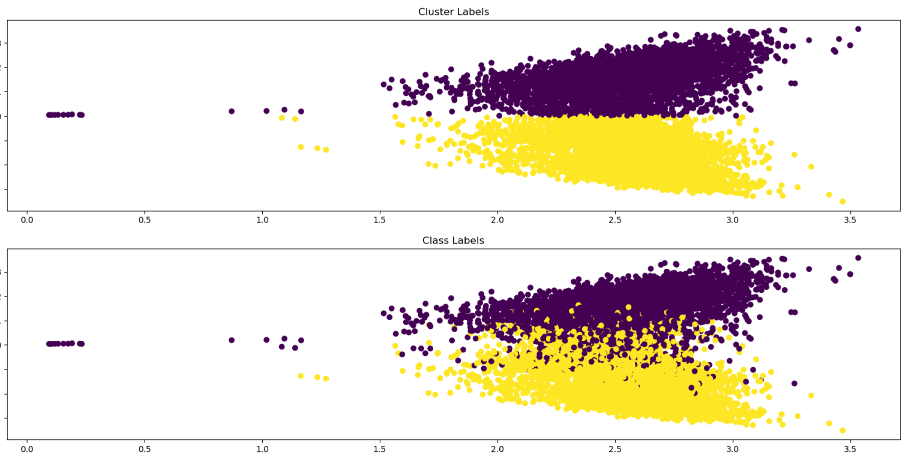


Figure 4: Logarithm transformation for SVD

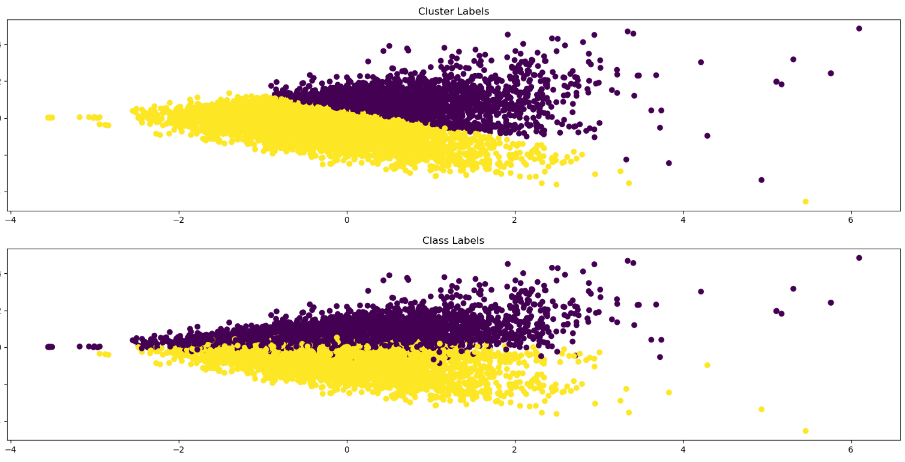


Figure 5: Mean and variance scaling for SVD

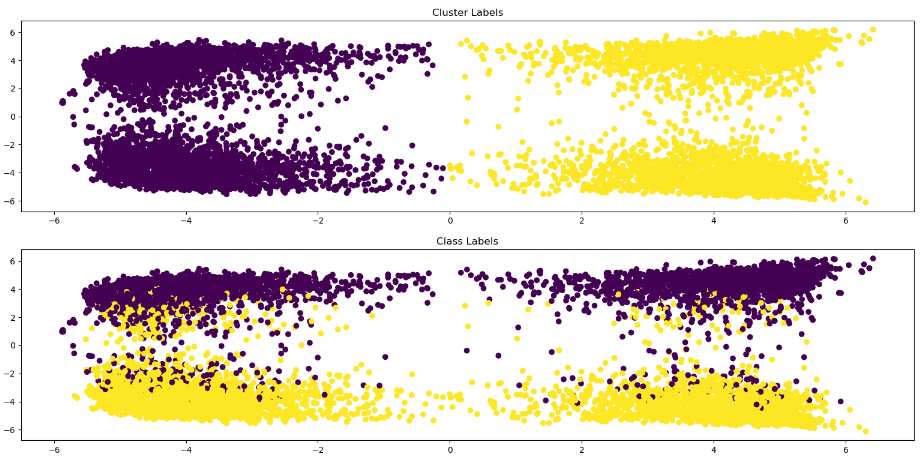


Figure 6: Mean and variance scaling followed by logarithm transformation for SVD

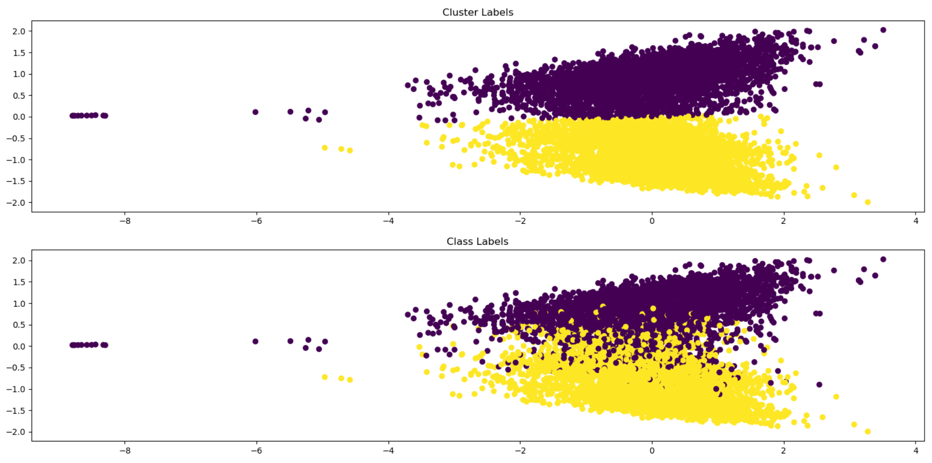


Figure 7: Logarithm transformation followed by mean and variance scaling for SVD

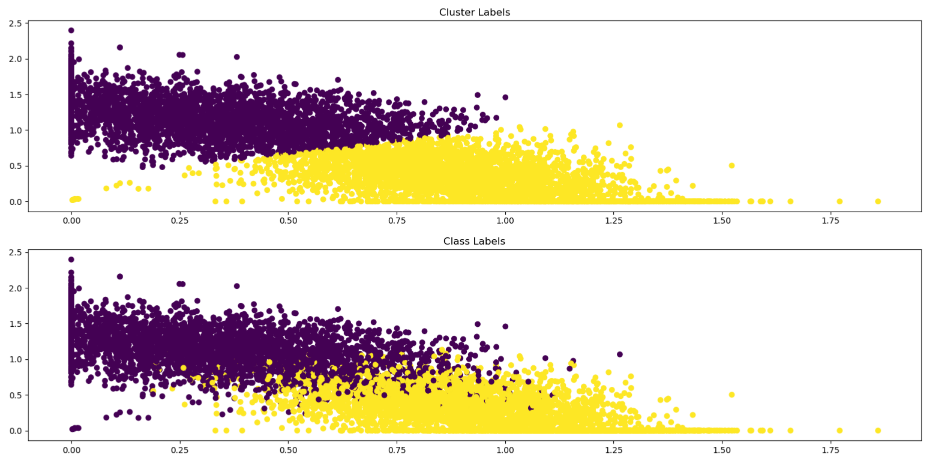


Figure 8: Logarithm transformation for NMF

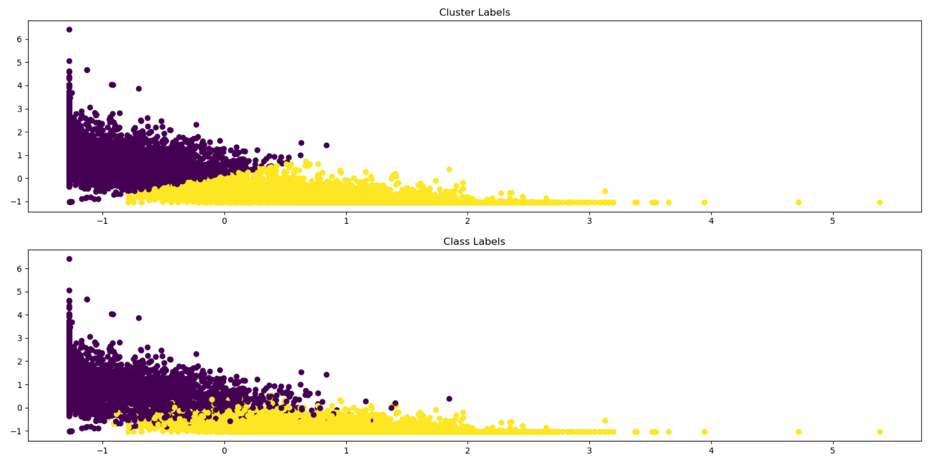


Figure 9: Mean and variance scaling for NMF

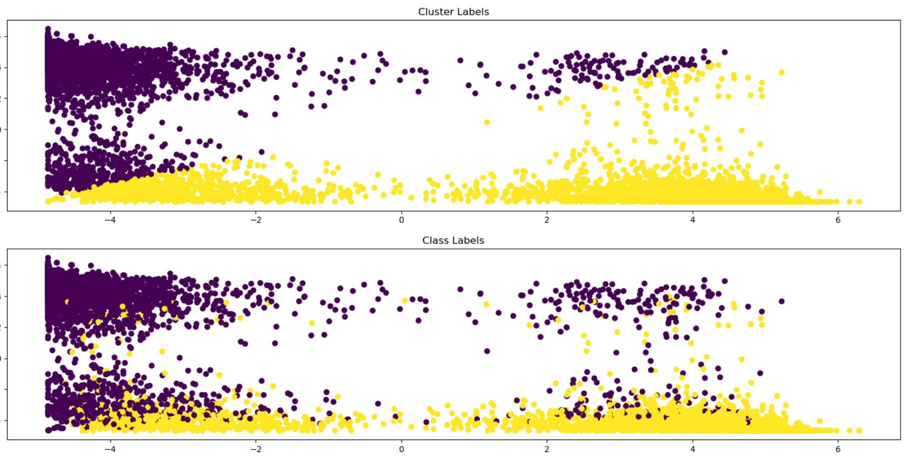


Figure 10: Mean and variance scaling followed by logarithm transformation for NMF

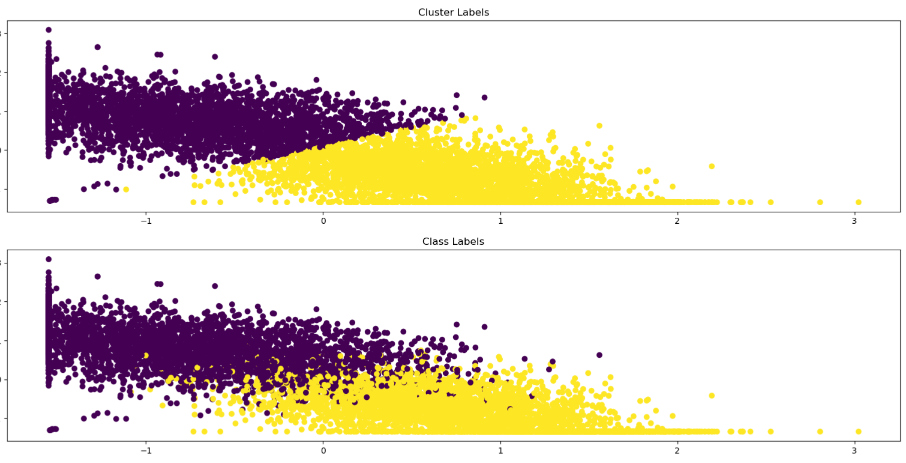


Figure 11: Logarithm transformation followed by mean and variance scaling for NMF

Question 9

The logarithm transformation increases the variability in feature descriptions for training samples, increasing differences between training points that are close together. It's important to note that the transformation does not further increase the separation of outliers, this is **due to the concavity of the *log* functions**. Smaller domain values into the *log* function result in more resolution for the image after the non-linear transformation; the resulting y is more isotropic to improve clustering performance.

Question 10

We now look at the metrics to determine the best transformation from the aforementioned combinations.

```
def scalingmetric(X, Target, c, best_r, scale=0, log=0, order=0, flag=
    False, n_cluster_flex=2):
    if order == 0:
        if scale == 0:
            X_1 = scalingfeatures(X)
        else:
            X_1 = X

        if log == 0:
            X_2 = logscale(X_1, c)
        else:
            X_2 = X_1

    else:
        X_1 = logscale(X, c)
        X_2 = scalingfeatures(X_1)

    kmeans = KMeans(n_clusters=n_cluster_flex, n_init=30, max_iter=1000,
     random_state=None).fit(
        X_2)
    pred = kmeans.predict(X_2)
    # print pred
    # print Bin_Target_Test

    if flag == True:
        print(print_metrics_and_return(Target, pred))
        plotKmeans(X_2, pred, Target, best_r)

    if flag == False:
        # print print_metrics_and_return(Bin_Target_Test, pred)
        return print_metrics_and_return(Target, pred)
    else:
        return 0, 0, 0, 0, 0
```

We got the following results for SVD and NMF:

		Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
SVD	Log	0.60941664	0.60940030	0.60938395	0.71650201	0.60934819
	Scale	0.23531901	0.24875208	0.26381164	0.25462540	0.23524900
	Scale then log	0.00007414	0.00007422	0.00007430	-0.00001320	-0.00001741
	Log then scale	0.60959478	0.60957446	0.60955415	0.71650201	0.60951841
NMF	Log	0.70089785	0.70154602	0.70219539	0.79501706	0.70087047
	Scale	0.68280383	0.68422195	0.68564598	0.77344268	0.68277479
	Scale then Log	0.69290883	0.69395077	0.69499584	0.78509197	0.69288072
	Log then scale	**0.70293308**	**0.70351202**	**0.70409191**	**0.79728146**	**0.70290589**

Table 3: Metrics for SVD and NMF with all possible combinations

From the table, we see that for NMF and SVD, logarithm transformation and logarithm transformation followed by mean and variance scaling works the best. Even intuitively, this would be the best course of action since we want to unskew the distributions of each feature before clustering. This effect of unskewing the data is maximized by applying the logarithm transformation on the full dynamic range compared to applying it on variables with a unit standard deviation.

Question 11

Reloading our data set with all 20 categories in the 20Newsgroups data set, we transform this to a TF-IDF matrix, with the intention of pipelining all the parameters we can chose to best cluster the data. The bag-of-words model generated the following dimensions: **X_tfidf size**: (18846, 52295) The following code snippet shows the **Naive** K-Means clustering on the 20 clusters.

```
kmeans_all = KMeans(n_clusters=20,  n_init=30, max_iter=1000,
    random_state=0).fit(X_tfidf_all)
pred_all = kmeans_all.predict(X_tfidf_all)

print(contingency_matrix(dataset_all_cat['target'], pred_all))
print(print_metrics_and_return(dataset_all_cat['target'], pred_all))
```

The output was generated as shown below: We first present the 5 measures yielded for Naive K-Means. We naturally expect the clustering to perform poorly. Below is the contingency matrix:

Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
0.35942083	0.40008032	0.45111242	0.13663614	0.35731879

Table 4: Performance of K-Means clustering algorithm

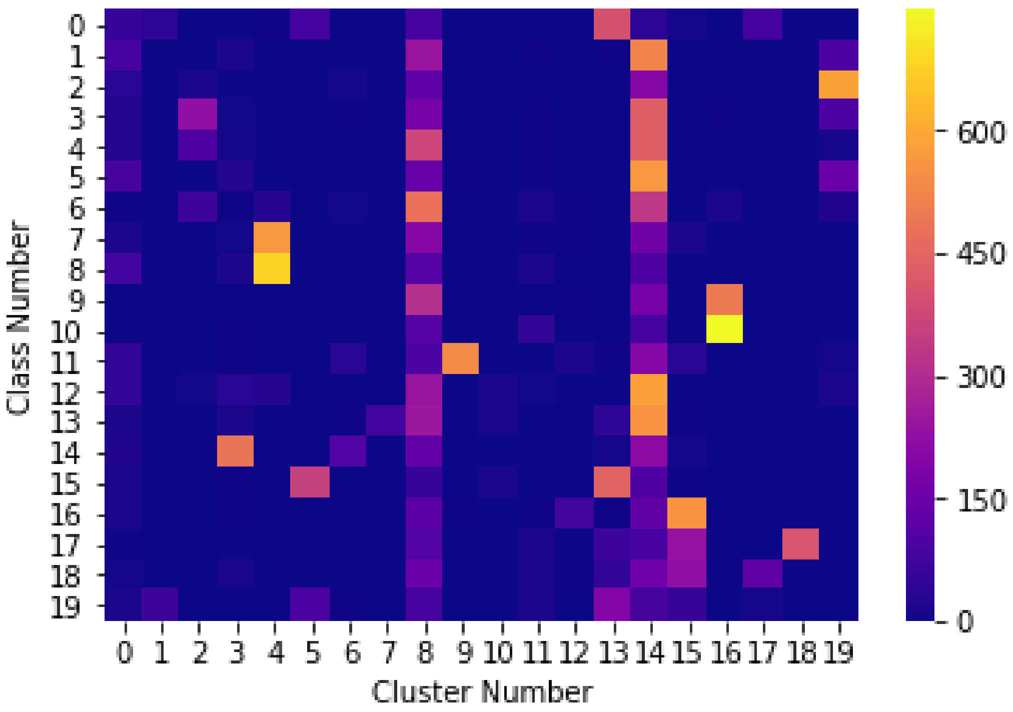


Figure 12: Contingency Matrix for Naive K-Means

Question 12

The trick here is to perform the scaling BEFORE we model the K-Means and find the resulting metrics. This way we first choose r, followed by SVD / NMF, and last we apply 5 transformations:

1. No transformations
2. Mean and variance scaling
3. Logarithm transformation
4. Mean and variance scaling followed by logarithm transformation
5. Logarithm transformation followed by mean and variance scaling

We also increased our range of r to incorporate higher values including $50, 100, 300, 500, 700$.

We designed the pipeline described above. We tested roughly 100 variations, including permutations in r, dimensionality reduction, and 5 variations of scaling. Key parts of the code to implement this is provided below:

```python
for i in [50,100,300,500,700]:
        print "Begin " + str(i) + "th regression..."
        # dimensionality reduction for all r
        t_svd12 = TruncatedSVD(n_components=i, n_iter=7, random_state
    =42)
        t_nmf12 = NMF(n_components=i, init='random', random_state=42)

        X_svd12 = t_svd12.fit_transform(X_tfidf)
        X_nmf12 = t_nmf12.fit_transform(X_tfidf)

        print "Researching best " + str(i) + "_SVM metrics: (Yeah!)"
        opt_metrics_svd = best_metrics_r_svd(X_svd12, Target, i)

        print "Researching best " + str(i) + "_NMF metrics: (Nah!)"
        opt_metrics_nmf = best_metrics_r_nmf(X_nmf12, Target, i)

        if opt_metrics_vmeas < opt_metrics_svd[1]:
            opt_metrics_vmeas = opt_metrics_svd[1]
            print "The new best parameters with " + str(i) + ", SVD"
            print opt_metrics_svd

        if opt_metrics_vmeas < opt_metrics_nmf[1]:
            opt_metrics_vmeas = opt_metrics_nmf[1]
            print "The new best parameters with " + str(i) + ", NMF"
            print opt_metrics_nmf
```

After many iterations, we obtained the best output as seen below in the terminal output. Note that the succeeding lines are those with the most recent optimal measures. We use V_measure again as described above. Below are sample metrics that our regression generated.

Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
0.35525951	0.37071384	0.38757390	0.18275702	0.35317490

Table 5: Settings for K-Mean Clustering: Number of Features = 50, Dimension Reduction = SVD, Transformation = Logarithmic Transform)

Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
0.47952220	0.51686839	0.56052309	0.25991087	0.47783237

Table 6: Settings for K-Mean Clustering: Number of Features = 100, Dimension Reduction = NMF, Transformation = Logarithmic Transform, followed by Scaling)

We observe that the first example, unlike the second, is not an ideal arrangement as the number of features are few, and we use SVD. Note that in part 8, we found NMF to be the better. Also the scaling was not done, and the regression chose to simply do a LOG transformation. Other inferior metrics follow a similar trend. The second metric on the other hand, follows more proven techniques and yields higher metrics.

The **best results** were obtained in the following table. Note that these results are in tune to the results derived in Question 5 and 8.

Homogeneity	V-measure	Completeness	Adjusted Rand Index	Adjusted MI Score
0.47789955	0.52206257	0.57521894	0.24848422	0.47619829

Table 7: Settings for K-Mean Clustering: Number of Features = 300, Dimension Reduction = NMF, Transformation = Logarithmic Transform, followed by Scaling)

It is observed that this result is much improved from the results obtained in Question 11. In fact, the **V_measure improves by** 30%, with other measures performing better as well. In comparison to the SVD classifier in table 5, the V_measure in the optimal setting is improved by 40%.

As a result, the 20-class clustering using a systematic hyper-parameter tuning resulted in improved metrics.