project2_final

February 7, 2022

1 Project 2: Data Representations and Clustering

Group Member Yifu Yuan, 804973353
Zhiquan You, 205667859
Wenxuan Wu, 705846929

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

[ ]:
```
from sklearn.datasets import fetch_20newsgroups

dataset = fetch_20newsgroups(subset='all', remove=('headers', 'footers'))
```

[ ]:
```
import numpy as np

subset_texts = []
subset_labels = []
total_labels = dataset.target_names
str_to_int_labels = {}
for idx, label in enumerate(total_labels):
    str_to_int_labels[label] = idx
class0 = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.
 ↪hardware', 'comp.sys.mac.hardware']
class1 = ['rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.
 ↪hockey']

class0_int = [str_to_int_labels[label] for label in class0]
class1_int = [str_to_int_labels[label] for label in class1]
for i in range(len(dataset.data)):
    if dataset.target[i] in class0_int:
        subset_texts.append(dataset.data[i])
        subset_labels.append(0)
    elif dataset.target[i] in class1_int:
        subset_texts.append(dataset.data[i])
        subset_labels.append(1)

subset_labels = np.array(subset_labels)
print(len(subset_texts))
```

7882

[ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

vectorizer = CountVectorizer(min_df=3, stop_words='english')
vectorizer.fit(subset_texts)
vocab = vectorizer.get_feature_names_out()

pipe = Pipeline([('count', CountVectorizer(vocabulary=vocab)),
                 ('tfid', TfidfTransformer())]).fit(subset_texts)
tf_idf_matrix = pipe.transform(subset_texts)
```

[ ]:
```
print(tf_idf_matrix.shape)
```

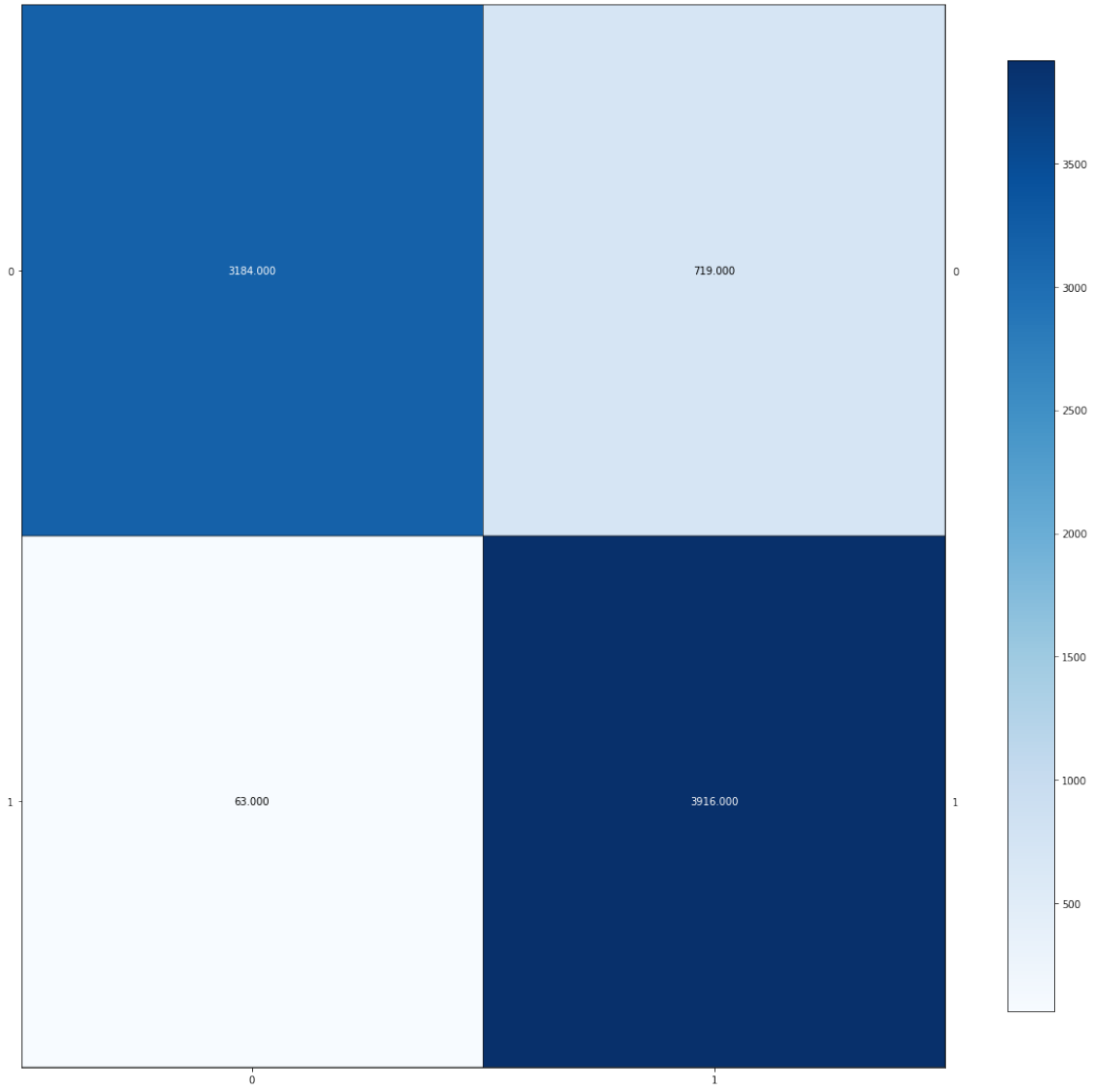
```
(7882, 23522)
```

As we can see, the shapes of the TF-IDF matrix is (7882, 19033).

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

[ ]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0, max_iter=1000, n_init=30).
 ↪fit(tf_idf_matrix)
kmeans_pred = kmeans.predict(tf_idf_matrix)
```

[ ]:
```
from plotmat import plot_mat
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix
from sklearn.metrics.cluster import contingency_matrix

labels = dataset.target
#kmeans_contingency_matrix = contingency_matrix(subset_labels, kmeans_pred)
#print(kmeans_contingency_matrix)
cm = confusion_matrix(subset_labels, kmeans_pred)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15))
```



The contingency matrix is reported as above. In a sense the matrix has to be square-shaped, because we would want the cluster number to match true class numbers. However, it is also feasible to have non-square-shaped contingency matrix.

1.0.3 QUESTION 3: Report the 5 clustering measures explained in the introduction for Kmeans clustering.

[ ]:
```
from sklearn import metrics

def report_five_scores(true_label, pred_label, verbose=False):
    homogeneity_score = metrics.homogeneity_score(true_label, pred_label)
    completeness_score = metrics.completeness_score(true_label, pred_label)
    v_measure_score = metrics.v_measure_score(true_label, pred_label)
```

```
    adjusted_rand_score = metrics.adjusted_rand_score(true_label, pred_label)
    adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(true_label,␣
↪pred_label)

    if verbose:
        print("homogeneity score is {}".format(metrics.
↪homogeneity_score(true_label, pred_label)))
        print("completeness score is {}".format(metrics.
↪completeness_score(true_label, pred_label)))
        print("v measure score is {}".format(metrics.
↪v_measure_score(true_label, pred_label)))
        print("adjusted rand score is {}".format(metrics.
↪adjusted_rand_score(true_label, pred_label)))
        print("adjusted mutual info score is {}".format(metrics.
↪adjusted_mutual_info_score(true_label, pred_label)))

    return homogeneity_score, completeness_score, v_measure_score,␣
↪adjusted_rand_score, adjusted_mutual_info_score

report_five_scores(subset_labels, kmeans_pred, verbose=True)
```

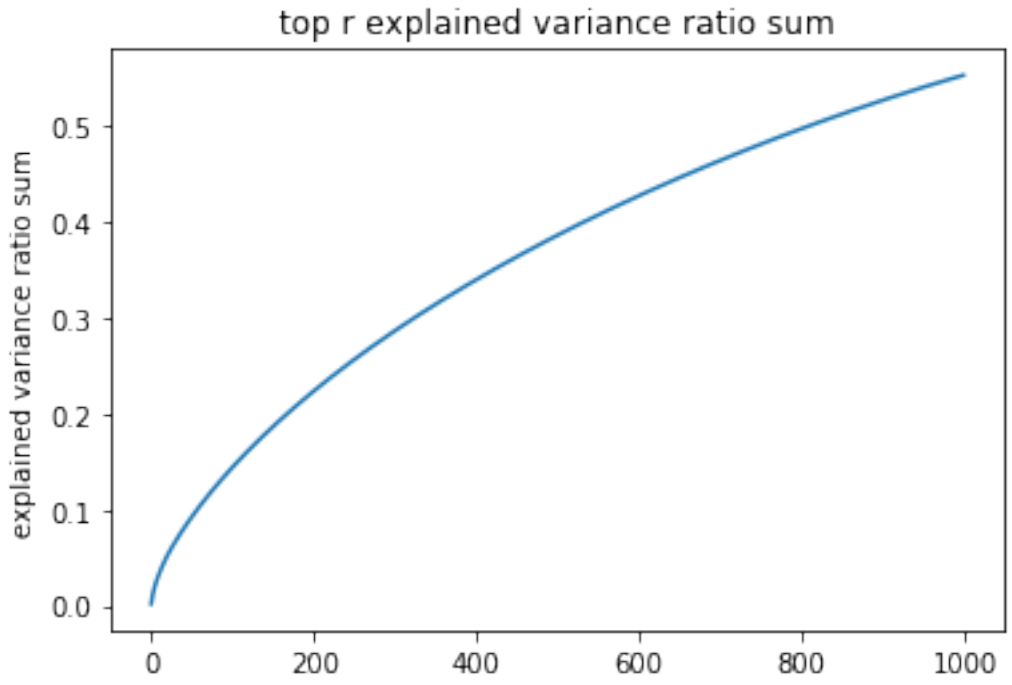
```
homogeneity score is 0.5770226536616643
completeness score is 0.5902566302706052
v measure score is 0.5835646221575521
adjusted rand score is 0.6424746172229008
adjusted mutual info score is 0.5835260647915069
```

[ ]:
```
(0.5770226536616643,
 0.5902566302706052,
 0.5835646221575521,
 0.6424746172229008,
 0.5835260647915069)
```

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

[ ]:
```
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt

svd = TruncatedSVD(n_components=1000)
svd.fit(tf_idf_matrix)

top_ev_sum = [svd.explained_variance_ratio_[0]]
for i in range(1, len(svd.explained_variance_ratio_)):
    top_ev_sum.append(svd.explained_variance_ratio_[i] + top_ev_sum[-1])
```

```
plt.plot(top_ev_sum)
plt.title('top r explained variance ratio sum')
plt.ylabel('explained variance ratio sum')
plt.show()
```



We can see that as the number of components r increases, the explained variance ratio increases as well.

1.0.5 QUESTION 5: Report a good choice of r for SVD and NMF respectively.

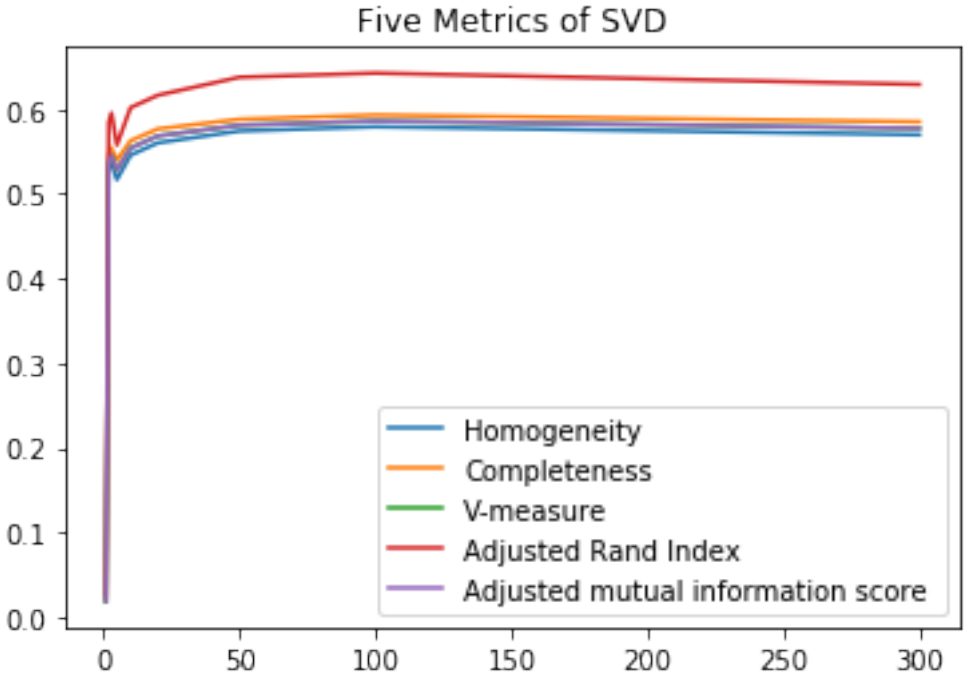
[ ]:
```
from sklearn.decomposition import NMF

svd_scores = []
nmf_scores = []
r_list = [1, 2, 3, 5, 10, 20, 50, 100, 300]
for r in r_list:
    svd_matrix = TruncatedSVD(n_components=r).fit_transform(tf_idf_matrix)
    nmf_matrix = NMF(n_components=r, tol=0.05, max_iter=50).
 ↪fit_transform(tf_idf_matrix)
    kmeans_svd = KMeans(n_clusters=2, random_state=0, max_iter=1500, n_init=30).
 ↪fit(svd_matrix)
    kmeans_nmf = KMeans(n_clusters=2, random_state=0, max_iter=1500, n_init=30).
 ↪fit(nmf_matrix)
    kmeans_svd_pred = kmeans_svd.predict(svd_matrix)
    kmeans_nmf_pred = kmeans_nmf.predict(nmf_matrix)
```

```
    svd_scores.append(report_five_scores(subset_labels, kmeans_svd_pred))
    nmf_scores.append(report_five_scores(subset_labels, kmeans_nmf_pred))

svd_scores = np.array(svd_scores)
nmf_scores = np.array(nmf_scores)
```

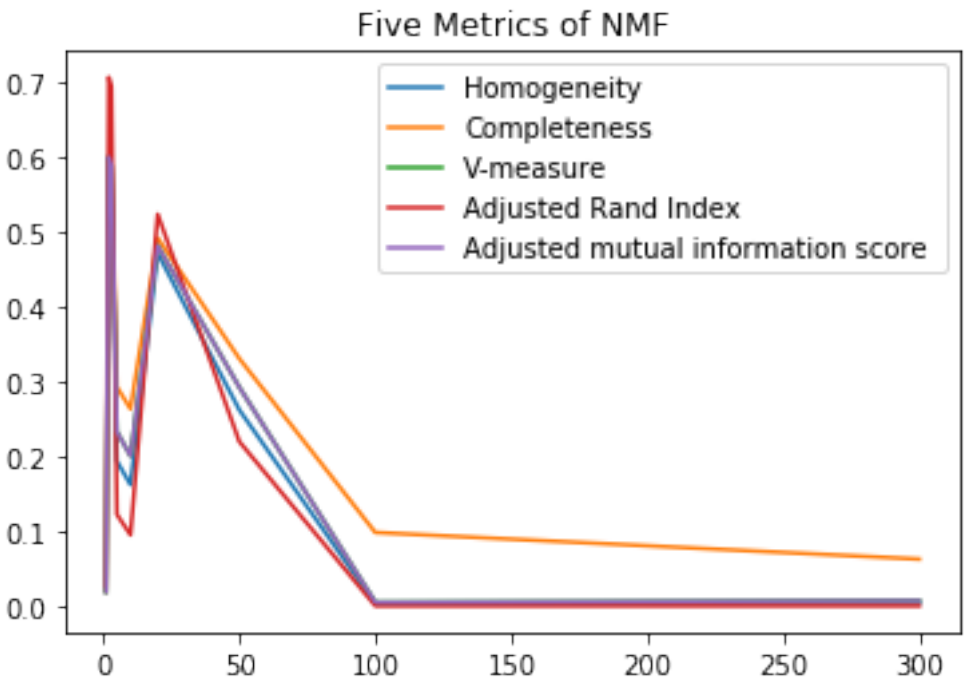
```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
```

```
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
```

[ ]:
```
plt.plot(r_list, svd_scores[:, 0], label="Homogeneity")
plt.plot(r_list, svd_scores[:, 1], label="Completeness")
plt.plot(r_list, svd_scores[:, 2], label="V-measure")
plt.plot(r_list, svd_scores[:, 3], label="Adjusted Rand Index")
plt.plot(r_list, svd_scores[:, 4], label="Adjusted mutual information score ")
plt.legend()
plt.title("Five Metrics of SVD")
plt.show()
```



[ ]:
```
plt.plot(r_list, nmf_scores[:, 0], label="Homogeneity")
plt.plot(r_list, nmf_scores[:, 1], label="Completeness")
plt.plot(r_list, nmf_scores[:, 2], label="V-measure")
plt.plot(r_list, nmf_scores[:, 3], label="Adjusted Rand Index")
plt.plot(r_list, nmf_scores[:, 4], label="Adjusted mutual information score ")
plt.legend()
plt.title("Five Metrics of NMF")
plt.show()
```



[ ]:
```
print(np.argmax(nmf_scores[:, 0]))
print(np.argmax(nmf_scores[:, 1]))
print(np.argmax(nmf_scores[:, 2]))
print(np.argmax(nmf_scores[:, 3]))
print(np.argmax(nmf_scores[:, 4]))

print(np.argmax(svd_scores[:, 0]))
print(np.argmax(svd_scores[:, 1]))
print(np.argmax(svd_scores[:, 2]))
print(np.argmax(svd_scores[:, 3]))
print(np.argmax(svd_scores[:, 4]))
```

```
1
1
1
1
1
7
7
7
7
7
```

[ ]:
```
svd_best_r = r_list[np.argmax(svd_scores[:, 4])]
nmf_best_r = r_list[np.argmax(nmf_scores[:, 4])]
```

```
print("Best r for SVD is {}".format(svd_best_r))
print("Best r for NMF is {}".format(nmf_best_r))
```

```
Best r for SVD is 100
Best r for NMF is 2
```

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

From both charts, we can observe the non-monotonic behaviors in the measures as the number of components increases. We can see that all the five measures initially increase or to the peak, then falls down and stays flat. Supposedly, the more the components, the more information would be preserved, and the better the clusters would be formed. However, it is not the case here with k-means clusters. The reason behind this might be that k-means clustering algorithms suffer from the curve of dimensionalities, as when the number of components(dimensions) increases, the data becomes more sparse in each dimension. And this causes the Euclidean distance to be no longer a good measure for forming clusters. In higher dimensions, all the distances between points will just be approaching 1. We can also observe that NMF chart changed drastically as compared to SVD chart. This might be due to the fact that NMF only allows positive vectors while SVD do not have this kind of restriction. Therefore, SVD has more information to feed in as the number of components increase till no more new information.

1.0.7 QUESTION 7: Are these measures on average better than those computed in Question 3?

1. For SVD, from the chart we can see that the 5 measures increased a little as compared to those computed in Q3, with the values over 0.6, as the number of dimensionalities increase.
2. For NMF, from the chart the measures peaked at the first few components and then drops tremendously, the average measures are worse than those computed in Q3, with the values drop down to between 0 and 0.1, as the number of dimensions increase.

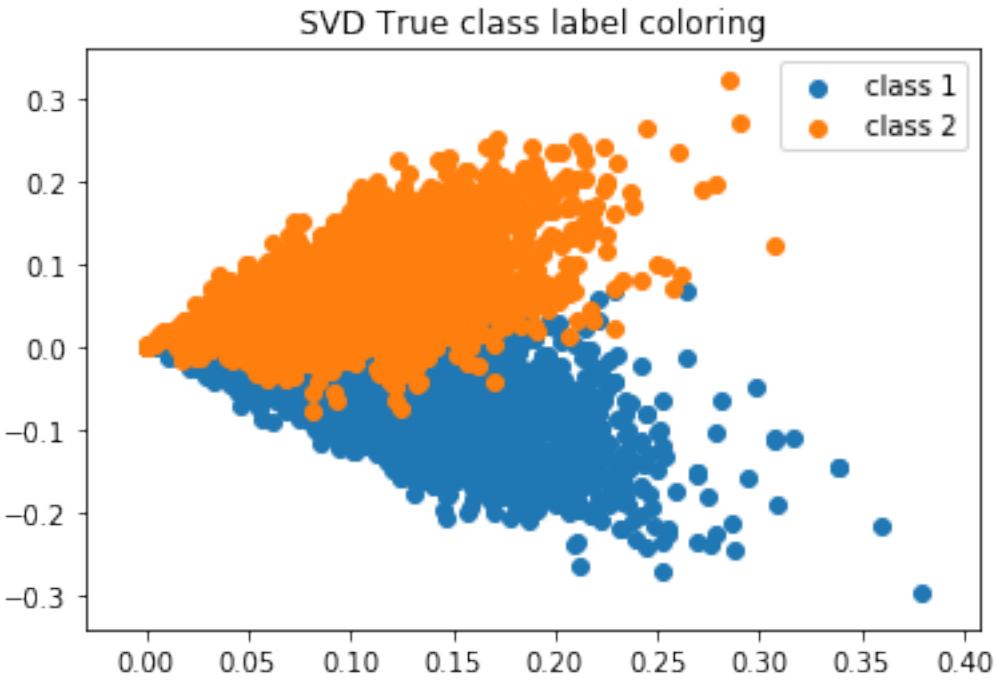
1.0.8 QUESTION 8: Visualize the clustering results

[ ]:
```
svd_matrix = TruncatedSVD(n_components=svd_best_r).fit_transform(tf_idf_matrix)
nmf_matrix = NMF(n_components=2).fit_transform(tf_idf_matrix)

svd_matrix_2d = TruncatedSVD(n_components=2).fit_transform(svd_matrix)
#nmf_matrix_2d = TruncatedSVD(n_components=2).fit_transform(nmf_matrix)
nmf_matrix_2d = nmf_matrix

kmeans_svd = KMeans(n_clusters=2, random_state=0, max_iter=1000, n_init=30).
 ↪fit(svd_matrix)
kmeans_nmf = KMeans(n_clusters=2, random_state=0, max_iter=1000, n_init=30).
 ↪fit(nmf_matrix)
kmeans_svd_pred = kmeans_svd.predict(svd_matrix)
```

```
kmeans_nmf_pred = kmeans_nmf.predict(nmf_matrix)
```

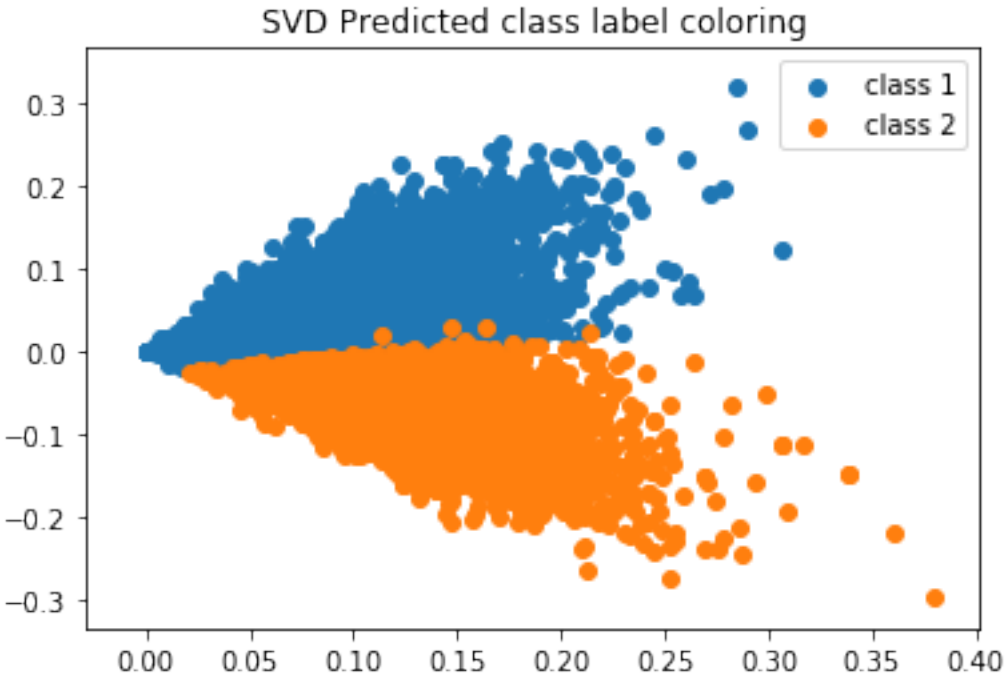
```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
```

[ ]:
```
svd_matrix_true_class0 = svd_matrix_2d[np.where(subset_labels==0)]
svd_matrix_true_class1 = svd_matrix_2d[np.where(subset_labels==1)]
plt.scatter(svd_matrix_true_class0[:, 0], svd_matrix_true_class0[:, 1],␣
 ↪label="class 1")
plt.scatter(svd_matrix_true_class1[:, 0], svd_matrix_true_class1[:, 1],␣
 ↪label="class 2")
plt.legend()
plt.title("SVD True class label coloring")
plt.show()
```

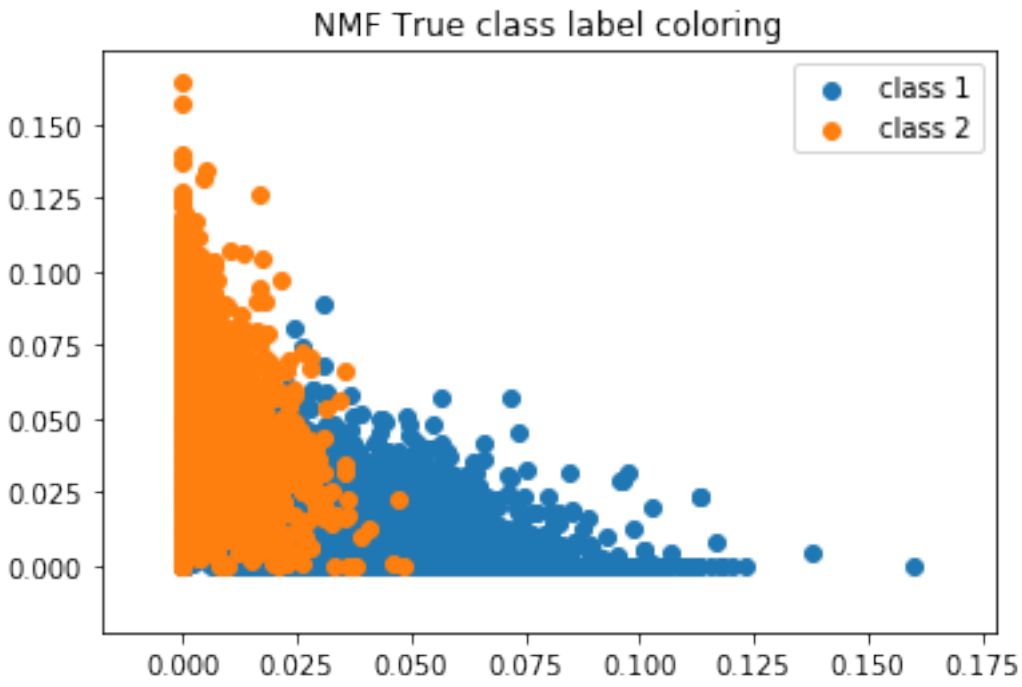


[ ]:
```
svd_matrix_pred_class0 = svd_matrix_2d[np.where(kmeans_svd_pred==0)]
svd_matrix_pred_class1 = svd_matrix_2d[np.where(kmeans_svd_pred==1)]
plt.scatter(svd_matrix_pred_class0[:, 0], svd_matrix_pred_class0[:, 1],␣
 ↪label="class 1")
plt.scatter(svd_matrix_pred_class1[:, 0], svd_matrix_pred_class1[:, 1],␣
 ↪label="class 2")
```

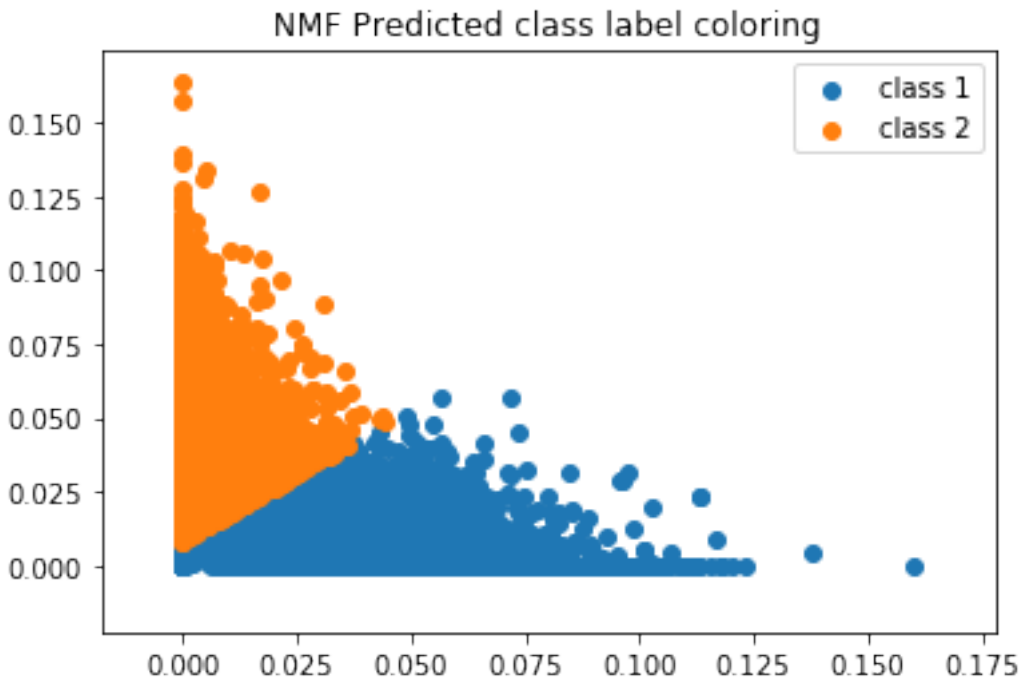
```
plt.legend()
plt.title("SVD Predicted class label coloring")
plt.show()
```



[ ]:
```
nmf_matrix_true_class0 = nmf_matrix_2d[np.where(subset_labels==0)]
nmf_matrix_true_class1 = nmf_matrix_2d[np.where(subset_labels==1)]
plt.scatter(nmf_matrix_true_class0[:, 0], nmf_matrix_true_class0[:, 1],␣
 ↪label="class 1")
plt.scatter(nmf_matrix_true_class1[:, 0], nmf_matrix_true_class1[:, 1],␣
 ↪label="class 2")
plt.legend()
plt.title("NMF True class label coloring")
plt.show()
```



[ ]:
```
nmf_matrix_pred_class0 = nmf_matrix_2d[np.where(kmeans_nmf_pred==0)]
nmf_matrix_pred_class1 = nmf_matrix_2d[np.where(kmeans_nmf_pred==1)]
plt.scatter(nmf_matrix_pred_class0[:, 0], nmf_matrix_pred_class0[:, 1],␣
 ↪label="class 1")
plt.scatter(nmf_matrix_pred_class1[:, 0], nmf_matrix_pred_class1[:, 1],␣
 ↪label="class 2")
plt.legend()
plt.title("NMF Predicted class label coloring")
plt.show()
```



1.0.9 QUESTION 9: What do you observe in the visualization? How are the data points of the two classes distributed? Is distribution of the data ideal for K-Means clustering?

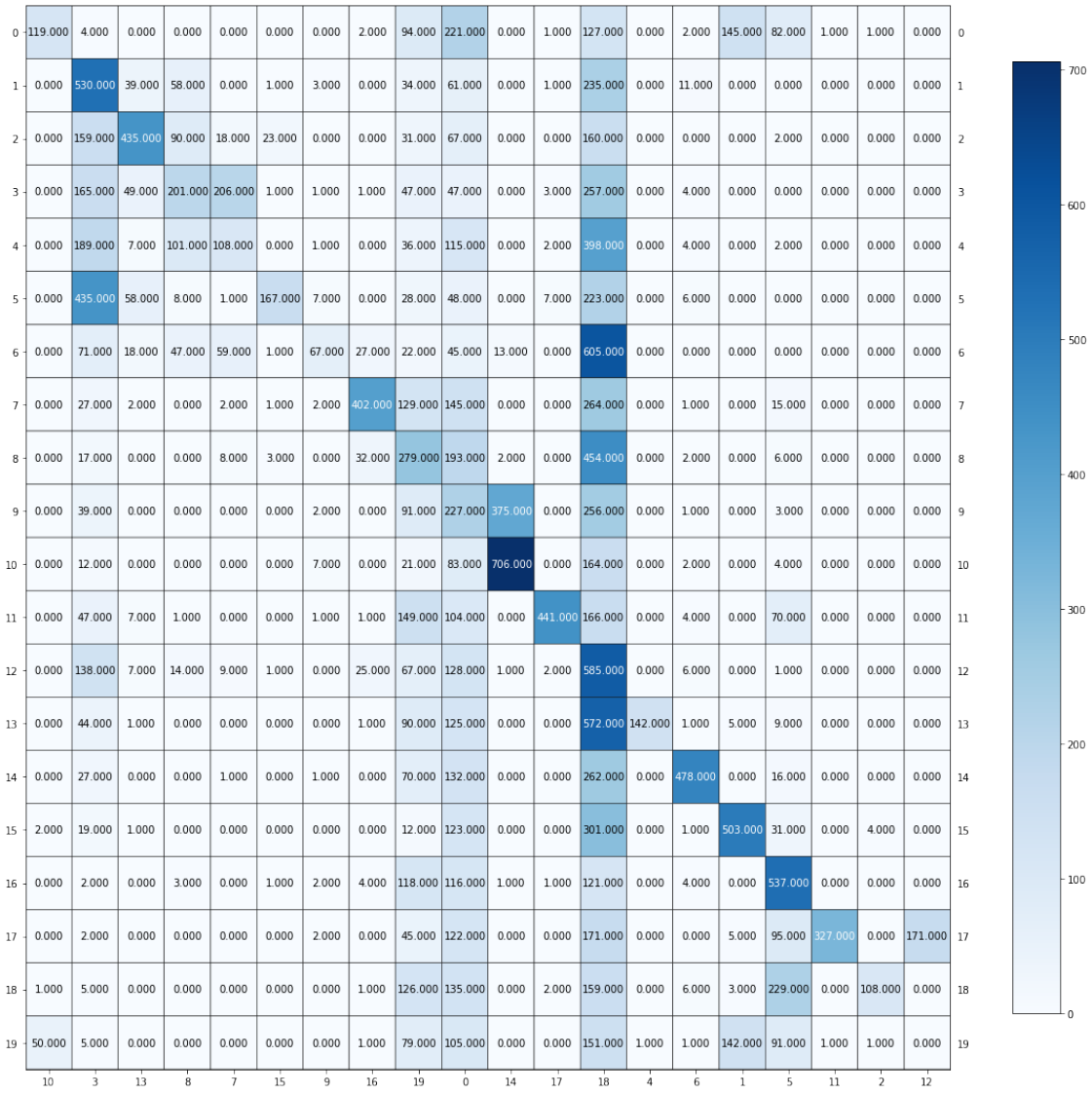
We can observe from the graphes that even though there are clusters forming, these clusters are not rounded (isotropic) as k-means implicitly assumed. Some of the data points in two clusters are very close to each other and even overlaps in two clusters(such as in the svd pred label graph). This makes the decision boundary vague. Also, K-means algorithm assumes that all variables have the same variance in nature. However, from our charts we can clearly see the uneven variance for data points in different classes. An ideal distribution of the data for K-means clustering would assume univariance across the dataset. Clearly, it is not the case here.

1.0.10 QUESTION 10: Visualize the contingency matrix and report the five clustering metrics.

[ ]:
```
texts = dataset.data
```

[ ]:
```
vectorizer = CountVectorizer(min_df=3, stop_words='english')
vectorizer.fit(texts)
vocab = vectorizer.get_feature_names_out()

pipe = Pipeline([('count', CountVectorizer(vocabulary=vocab)),
                 ('tfid', TfidfTransformer())]).fit(texts)
total_tf_idf_matrix = pipe.transform(texts)

svd = TruncatedSVD(n_components=100)
svd_total_matrix = svd.fit_transform(total_tf_idf_matrix)
```

[ ]:
```
kmeans = KMeans(n_clusters=20, random_state=0, max_iter=1000, n_init=30).
 ↪fit(svd_total_matrix)
kmeans_pred = kmeans.predict(svd_total_matrix)

labels = dataset.target
cm = confusion_matrix(labels, kmeans_pred)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15))
```



[ ]:
```
report_five_scores(labels, kmeans_pred, verbose=True)
```

```
homogeneity score is 0.31695950852156357
completeness score is 0.3919670246868399
```

```
v measure score is 0.3504952055867773
adjusted rand score is 0.10183006928976894
adjusted mutual info score is 0.34815436677239925
```

[ ]:
```
(0.31695950852156357,
 0.3919670246868399,
 0.3504952055867773,
 0.10183006928976894,
 0.34815436677239925)
```

1.0.11 Question 11 Use UMAP to reduce the dimensionality of the 20 category TF-IDF matrix, and apply K-Means clustering with n_components=20. Find a good n components choice for UMAP, and compare the performance of two metrics by setting metric="euclidean" and metric="cosine" respectively.

[ ]:
```
import umap.umap_ as umap
u_map_components = [5,20,200]

u_cos_score = []
u_euc_score = []

for r in u_map_components:
  print(r)
  reducer_euc = umap.UMAP(n_components=r, metric="euclidean")
  umap_matrix_euc = reducer_euc.fit_transform(total_tf_idf_matrix)
  kmeans_euc = KMeans(n_clusters=20, random_state=0, max_iter=1000, n_init=30).
 ↪fit(umap_matrix_euc)
  kmeans_pred_euc = kmeans_euc.predict(umap_matrix_euc)
  u_euc_score.append(report_five_scores(labels, kmeans_pred_euc))

  reducer_cos = umap.UMAP(n_components=r, metric="cosine")
  umap_matrix_cos = reducer_cos.fit_transform(total_tf_idf_matrix)
  kmeans_cos = KMeans(n_clusters=20, random_state=0, max_iter=1000, n_init=30).
 ↪fit(umap_matrix_cos)
  kmeans_pred_cos = kmeans_cos.predict(umap_matrix_cos)
  u_cos_score.append(report_five_scores(labels, kmeans_pred_cos))
```

```
5
20
200
```

[ ]:
```
u_euc_score = np.array(u_euc_score)

print(u_euc_score)
avg_metrics=[]
for measure in zip(u_euc_score[:, 0], u_euc_score[:, 1], u_euc_score[:, 2],␣
 ↪u_euc_score[:, 3], u_euc_score[:, 4]):
```

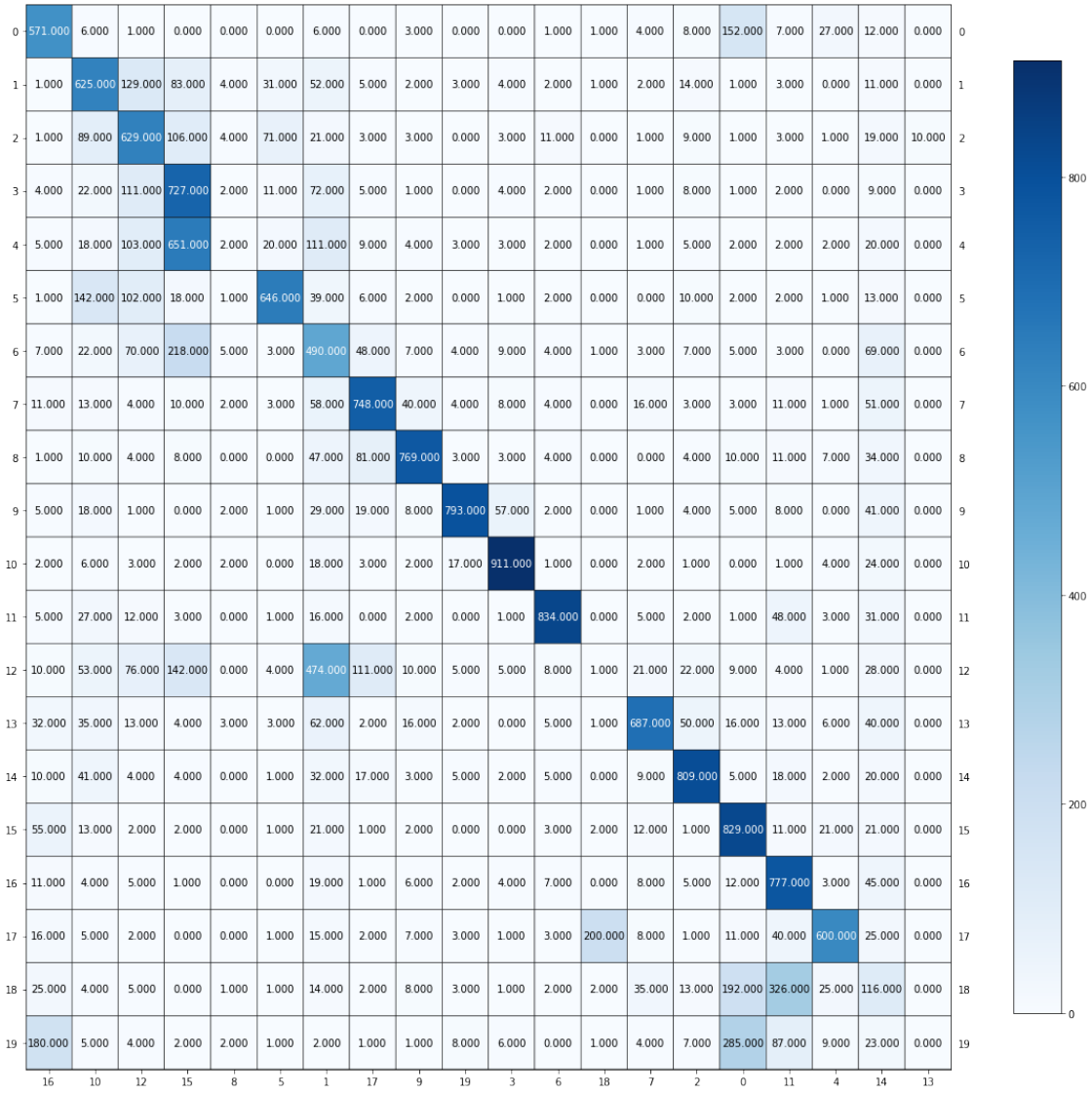
```
  avg_metrics.append(sum(measure) / 5)

best_euc_r = u_map_components[avg_metrics.index(max(avg_metrics))]

print(best_euc_r)
```

```
[[0.00915417 0.00962597 0.00938414 0.00089171 0.00610916]
 [0.00800148 0.00836436 0.0081789  0.00085295 0.0048879 ]
 [0.00760892 0.00810823 0.00785064 0.00065883 0.00456615]]
5
```

[ ]:
```
u_cos_score = np.array(u_cos_score)

print(u_cos_score)
avg_metrics=[]
for measure in zip(u_cos_score[:, 0], u_cos_score[:, 1], u_cos_score[:, 2],␣
 ↪u_cos_score[:, 3], u_cos_score[:, 4]):
  avg_metrics.append(sum(measure) / 5)
best_cos_r = u_map_components[avg_metrics.index(max(avg_metrics))]
print(best_cos_r)
```

```
[[0.57534995 0.59934852 0.58710409 0.46083518 0.58573385]
 [0.58197121 0.59684902 0.58931623 0.45894032 0.58797262]
 [0.57212806 0.59792233 0.58474087 0.44743219 0.58335811]]
20
```

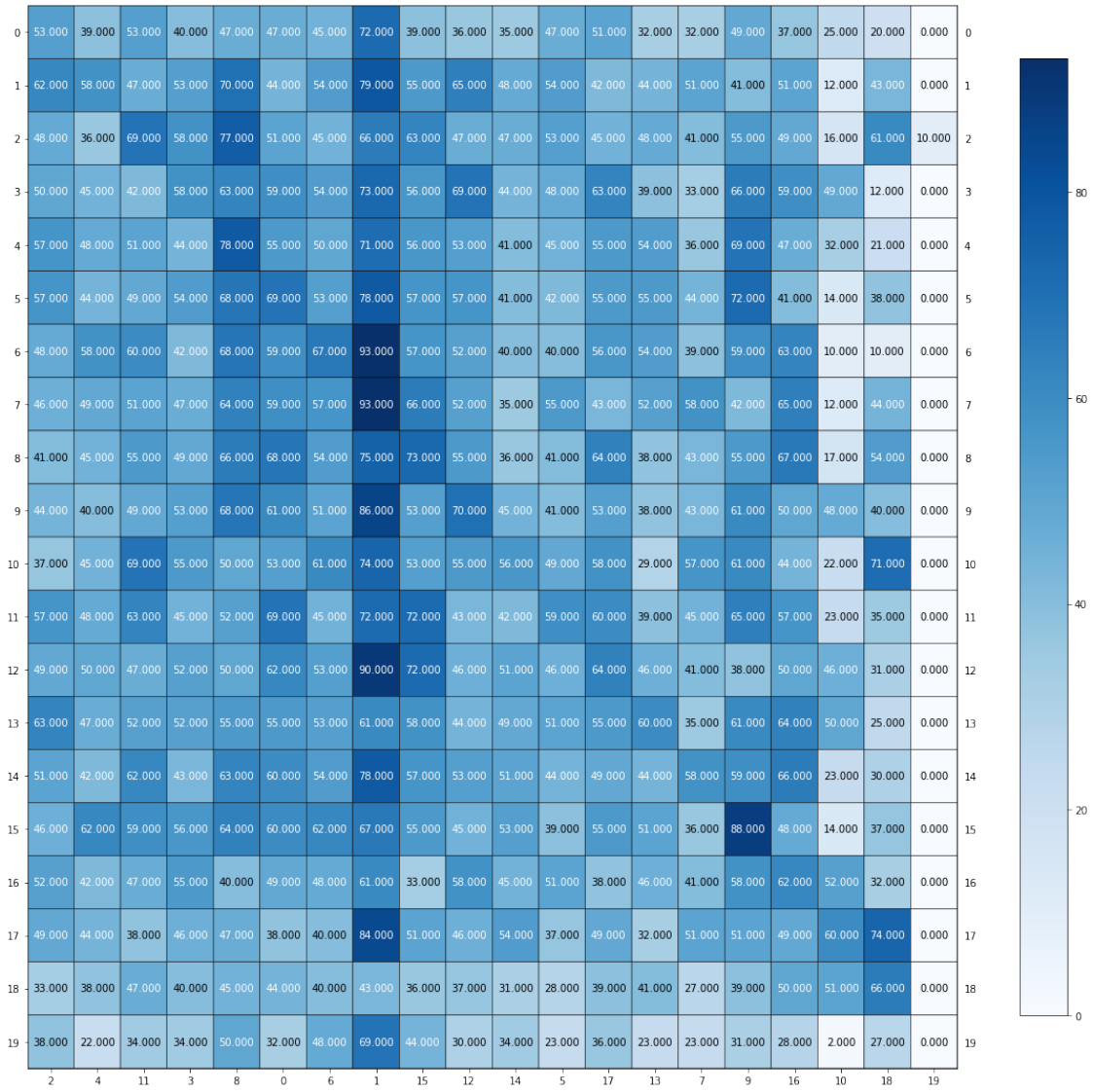
[ ]:
```
# Plot and scores for cosine

best_umap_cos = umap.UMAP(n_components=best_cos_r, metric="cosine").
 ↪fit_transform(total_tf_idf_matrix)
best_kmeans_cos = KMeans(n_clusters=20, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit(best_umap_cos)
kmeans_pred_cos = best_kmeans_cos.predict(best_umap_cos)

cm = confusion_matrix(labels, kmeans_pred_cos)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15))

_ = report_five_scores(labels, kmeans_pred_cos, verbose=True)
```




```
homogeneity score is 0.5661412970904304
completeness score is 0.5988095044573374
v measure score is 0.5820173506265517
adjusted rand score is 0.4474480005451306
adjusted mutual info score is 0.5806161766052441
```

[ ]:
```
# Plot and scores for euclidean

best_umap_euc = umap.UMAP(n_components=best_euc_r, metric="euclidean").
 ↪fit_transform(total_tf_idf_matrix)
best_kmeans_euc = KMeans(n_clusters=20, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit(best_umap_euc)
kmeans_pred_euc = best_kmeans_euc.predict(best_umap_euc)
```

```
cm = confusion_matrix(labels, kmeans_pred_euc)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15))

_ = report_five_scores(labels, kmeans_pred_euc, verbose=True)
```




```
homogeneity score is 0.007553748929275003
completeness score is 0.007707180437614607
v measure score is 0.007629693392680481
adjusted rand score is 0.0009439801610259739
adjusted mutual info score is 0.004391566603415189
```

1.0.12 QUESTION 12: Analyze the contingency matrix.

From the contingency matrix of both UMAPs with different distance metrics as well as the five measurement metrics, we can see that UMAP with cosine distance metric perform way better as compared to UMAP with Euclidean distance metrics. The reason behind it is that cosine distance is not affected by the magnitude of the vectors as well as the dimensionalities. In higher dimensions, the data becomes sparse and the Euclidean distance does not perform well under this condition; the distance between each data points would just be very similar to each other. The model can't seem to form obvious clusters under the condition that the feature points are all equidistant. The contingency matrix for the Euclidean distance is then highly non-diagonal and does not provide much help for data analysis.

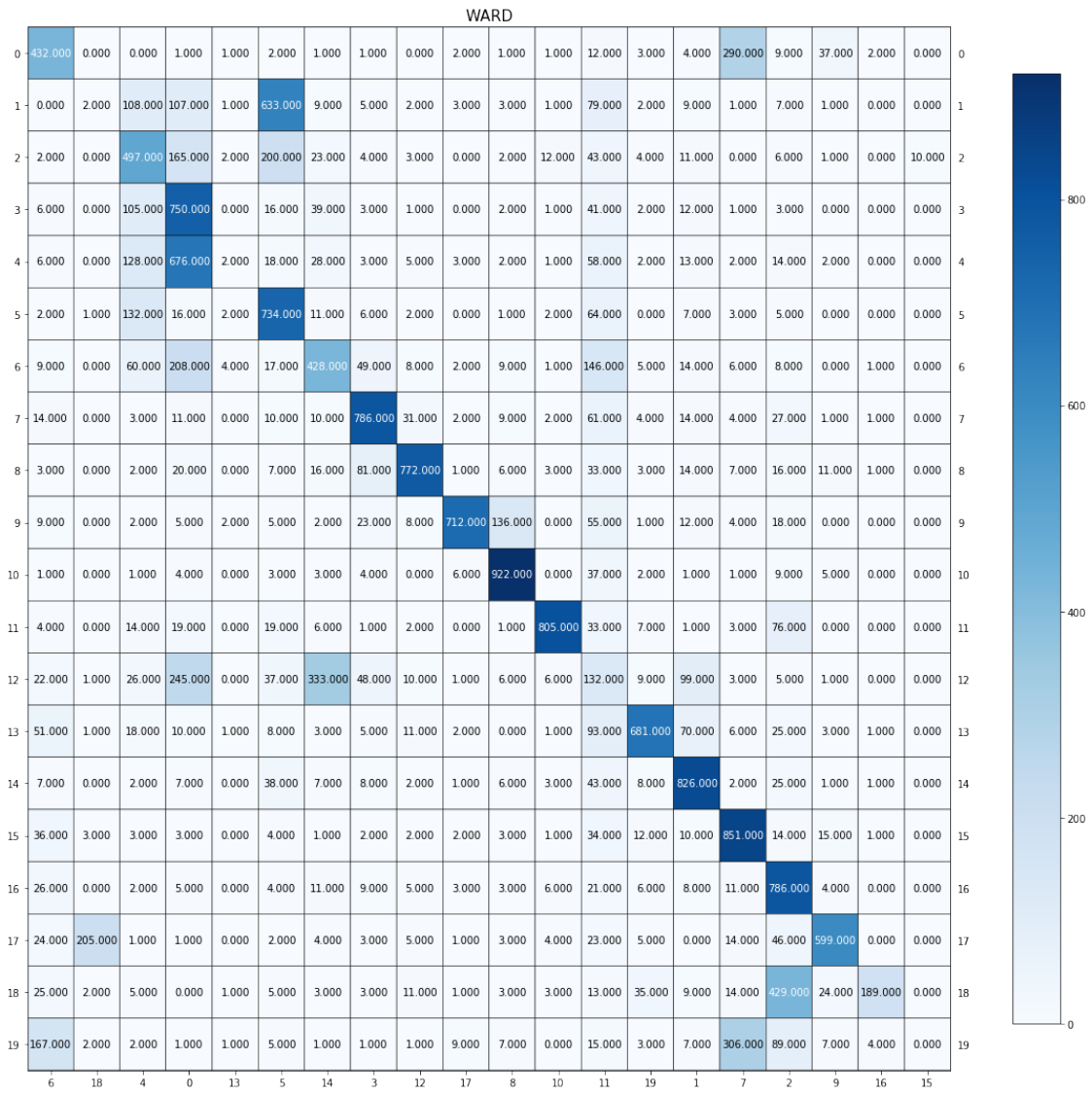
1.0.13 QUESTION 13: So far, we have attempted K-Means clustering with 4 different representation learning techniques (sparse representation, PCA, NMF, UMAP). Compare and contrast the results from the previous sections, and discuss which approach is best for the K-Means clustering task on the 20-class text data.

From Q10's results, we can see that K-means clustering with SVD(or PCA) perform very badly when the number of categories are 20, with each of the five measures near 0.4. From Q5's result, we can see that NMF's performance metrics decrease tremendouly as the number of dimensions increases. Even though from Q3's results we can observe that K-means with sparse representation has each metrics near 0.55, it just has 2 classes. Undoutedly, when the number of classes increase, the performance will decrease. Lastly, we can observe from Q11, UMAP with cosine distance perform very well with K-Means clustering with 20-class text data, with the metrics also around 0.55. Therefore, we think K-Means clustering with UMAP is the best for the task on the 20-class text data.

1.0.14 QUESTION 14: Report the five clustering evaluation metrics for Agglomerative Clustering ("ward" and "single")

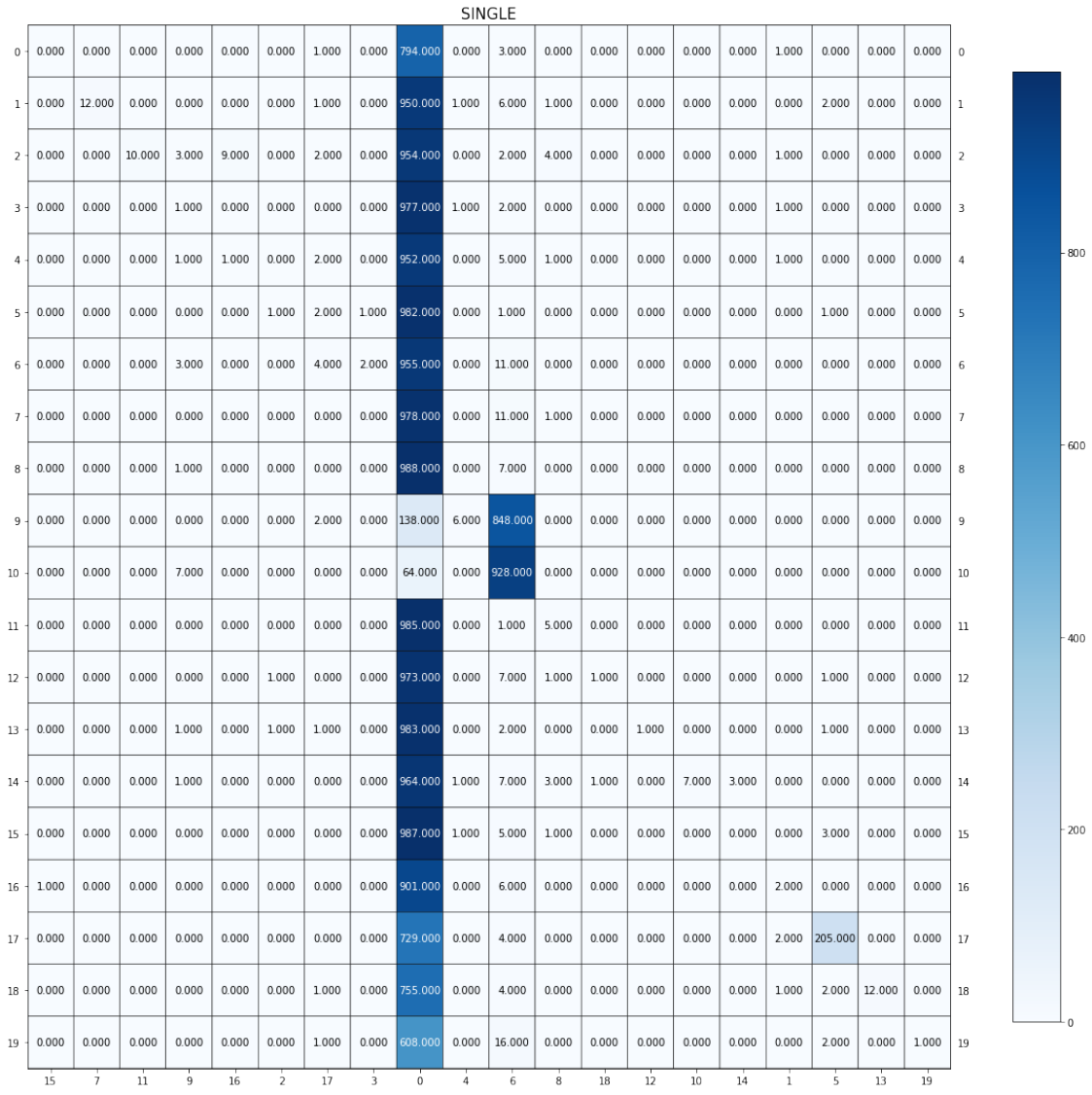
[ ]:
```
from sklearn.cluster import AgglomerativeClustering
ward_pred = AgglomerativeClustering(n_clusters=20, linkage='ward').
 ↪fit_predict(umap_matrix_cos)
single_pred = AgglomerativeClustering(n_clusters=20, linkage='single').
 ↪fit_predict(umap_matrix_cos)
```

[ ]:
```
# Plot and scores for ward
cm = confusion_matrix(labels, ward_pred)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], title='WARD', xticklabels=cols,␣
 ↪yticklabels=rows, size=(15,15))

_ = report_five_scores(labels, ward_pred, verbose=True)
```




```
homogeneity score is 0.559758558966435
completeness score is 0.6014490492687404
v measure score is 0.579855403500282
adjusted rand score is 0.42554145662581194
adjusted mutual info score is 0.5784369379572434
```

[ ]:
```
# Plot and scores for single
cm = confusion_matrix(labels, single_pred)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], title='SINGLE', xticklabels=cols,␣
 ↪yticklabels=rows, size=(15,15))

_ = report_five_scores(labels, single_pred, verbose=True)
```




```
homogeneity score is 0.1015466904708247
completeness score is 0.6808035154718326
v measure score is 0.1767324743623437
adjusted rand score is 0.020049499239432972
adjusted mutual info score is 0.17287019902306847
```

The performance for the ward linkage:
homogeneity score is 0.559758558966435
completeness score is 0.6014490492687404
v measure score is 0.579855403500282
adjusted rand score is 0.42554145662581194
adjusted mutual info score is 0.5784369379572434

The performance for the single linkage:
homogeneity score is 0.1015466904708247

completeness score is 0.6808035154718326
v measure score is 0.1767324743623437
adjusted rand score is 0.020049499239432972
adjusted mutual info score is 0.17287019902306847

As shown above, the ward linkage outperforms single linkage.

1.0.15 QUESTION 15: Experiment on DBSCAN and HDBSCAN

Since TA Arash Vahabpour in the post https://piazza.com/class/kxy3hn64g767gl?cid=126 pointed out that we do not need to perform hyperparameter tuning in Q15, we skipped the tuning part.

[ ]:
```
def printResults(verbose, tune_db_hdb_best, db_score, hdb_score, svd_score,␣
↪nmf_score, umap_score, autoencoder_score, hyper_param, cluster,␣
↪withDimReduc, wo_dr_score, grid_HDBSCAN):

   if tune_db_hdb_best == True:
       #DBSCAN has two hyperparameters like eps & min_samples.

       db_score_list = np.array(db_score)
       hdb_score_list = np.array(hdb_score)

       db_homo = db_score_list[:,0]
       db_cm = db_score_list[:,1]
       db_vm = db_score_list[:,2]
       db_ar = db_score_list[:,3]
       db_ami = db_score_list[:,4]

       hdb_homo = hdb_score_list[:,0]
       hdb_cm = hdb_score_list[:,1]
       hdb_vm = hdb_score_list[:,2]
       hdb_ar = hdb_score_list[:,3]
       hdb_ami = hdb_score_list[:,4]

       db_avg_metrics = []
       hdb_avg_metrics = []

       for measure in zip(db_homo, db_cm, db_vm, db_ar, db_ami):
           db_avg_metrics.append(sum(measure) / 5)

       for measure in zip(hdb_homo, hdb_cm, hdb_vm, hdb_ar, hdb_ami):
           hdb_avg_metrics.append(sum(measure) / 5)

       if verbose == True:

         print('DBSCAN Homogeneity: ', db_homo[db_avg_metrics.
↪index(max(db_avg_metrics))])
```

```
        print('DBSCAN Completeness: ', db_cm[db_avg_metrics.
↪index(max(db_avg_metrics))])
        print('DBSCAN V-measure: ', db_vm[db_avg_metrics.
↪index(max(db_avg_metrics))])
        print('DBSCAN Adjusted Random Index: ', db_ar[db_avg_metrics.
↪index(max(db_avg_metrics))])
        print('DBSCAN Adjusted Mutual Information Score: ',␣
↪db_ami[db_avg_metrics.index(max(db_avg_metrics))])

       ␣
↪print('-------------------------------------------------------------------------')

        print('HDBSCAN Homogeneity: ', hdb_homo[hdb_avg_metrics.
↪index(max(hdb_avg_metrics))])
        print('HDBSCAN Completeness: ', hdb_cm[hdb_avg_metrics.
↪index(max(hdb_avg_metrics))])
        print('HDBSCAN V-measure: ', hdb_vm[hdb_avg_metrics.
↪index(max(hdb_avg_metrics))])
        print('HDBSCAN Adjusted Random Index: ', hdb_ar[hdb_avg_metrics.
↪index(max(hdb_avg_metrics))])
        print('HDBSCAN Adjusted Mutual Information Score: ',␣
↪hdb_ami[hdb_avg_metrics.index(max(hdb_avg_metrics))])


  if withDimReduc == True and tune_db_hdb_best == False:
      svd_score = np.array(svd_score)
      nmf_score= np.array(nmf_score)
      umap_score= np.array(umap_score)
      autoencoder_score = np.array(autoencoder_score)

      svd_avg_metrics = []
      nmf_avg_metrics = []
      umap_avg_metrics = []
      autoencoder_avg_metrics = []

      if len(svd_score) > 0:
          svd_homo = svd_score[:,0]
          svd_cm = svd_score[:,1]
          svd_vm = svd_score[:,2]
          svd_ar = svd_score[:,3]
          svd_ami = svd_score[:,4]

          for measure in zip(svd_homo, svd_cm, svd_vm, svd_ar, svd_ami):
              svd_avg_metrics.append(sum(measure) / 5)

          if verbose == True:
```

```
            if grid_HDBSCAN == False:
                print('SVD Component: ', hyper_param[svd_avg_metrics.
↪index(max(svd_avg_metrics))][0], cluster, ': ', hyper_param[svd_avg_metrics.
↪index(max(svd_avg_metrics))][1])
                print("SVD Best Homogeneity: ", svd_homo[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best Completeness: ", svd_cm[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best V measure: ", svd_vm[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best Adjusted Rand Index: ",␣
↪svd_ar[svd_avg_metrics.index(max(svd_avg_metrics))])
                print("SVD Best Adjusted Mutual Info Score: ",␣
↪svd_ami[svd_avg_metrics.index(max(svd_avg_metrics))])

                ␣
↪print("-----------------------------------------------------------------------------------------

            else:
                print(cluster, 'SVD Component 50, Best min cluster size is:␣
↪', hyper_param[svd_avg_metrics.index(max(svd_avg_metrics))][0] , ' Best min␣
↪sample size is: ', hyper_param[svd_avg_metrics.
↪index(max(svd_avg_metrics))][1])
                print("SVD Best Homogeneity: ", svd_homo[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best Completeness: ", svd_cm[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best V measure: ", svd_vm[svd_avg_metrics.
↪index(max(svd_avg_metrics))])
                print("SVD Best Adjusted Rand Index: ",␣
↪svd_ar[svd_avg_metrics.index(max(svd_avg_metrics))])
                print("SVD Best Adjusted Mutual Info Score: ",␣
↪svd_ami[svd_avg_metrics.index(max(svd_avg_metrics))])

                ␣
↪print("-----------------------------------------------------------------------------------------

        if len(nmf_score) > 0:
            nmf_homo = nmf_score[:,0]
            nmf_cm = nmf_score[:,1]
            nmf_vm = nmf_score[:,2]
            nmf_ar = nmf_score[:,3]
            nmf_ami = nmf_score[:,4]

            for measure in zip(nmf_homo, nmf_cm, nmf_vm, nmf_ar, nmf_ami):
                nmf_avg_metrics.append(sum(measure) / 5)
```

```
            if verbose == True:
              if grid_HDBSCAN == False:
                    print('NMF Component: ', hyper_param[nmf_avg_metrics.
↪index(max(nmf_avg_metrics))][0], cluster, ': ',  hyper_param[nmf_avg_metrics.
↪index(max(nmf_avg_metrics))][1])
                    print("NMF Best Homogeneity: ", nmf_homo[nmf_avg_metrics.
↪index(max(nmf_avg_metrics))])
                    print("NMF Best Completeness: ", nmf_cm[nmf_avg_metrics.
↪index(max(nmf_avg_metrics))])
                    print("NMF Best V measure: ", nmf_vm[nmf_avg_metrics.
↪index(max(nmf_avg_metrics))])
                    print("NMF Best Adjusted Rand Index: ",␣
↪nmf_ar[nmf_avg_metrics.index(max(nmf_avg_metrics))])
                    print("NMF Best Adjusted Mutual Info Score: ",␣
↪nmf_ami[nmf_avg_metrics.index(max(nmf_avg_metrics))])

                  ␣
↪print("-------------------------------------------------------------------------------------


        if len(umap_score) > 0:
            umap_homo = umap_score[:,0]
            umap_cm = umap_score[:,1]
            umap_vm = umap_score[:,2]
            umap_ar = umap_score[:,3]
            umap_ami = umap_score[:,4]

            for measure in zip(umap_homo, umap_cm, umap_vm, umap_ar, umap_ami):
                umap_avg_metrics.append(sum(measure) / 5)

            if verbose == True:
              if grid_HDBSCAN == False:
                    print('UMAP Component: ', hyper_param[umap_avg_metrics.
↪index(max(umap_avg_metrics))][0], cluster, ': ', ␣
↪hyper_param[umap_avg_metrics.index(max(umap_avg_metrics))][1])
                    print("UMAP Best Homogeneity: ", umap_homo[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                    print("UMAP Best Completeness: ", umap_cm[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                    print("UMAP Best V measure: ", umap_vm[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                    print("UMAP Best Adjusted Rand Index: ",␣
↪umap_ar[umap_avg_metrics.index(max(umap_avg_metrics))])
```

```
                print("UMAP Best Adjusted Mutual Info Score: ",␣
↪umap_ami[umap_avg_metrics.index(max(umap_avg_metrics))])

                ␣
↪print("-------------------------------------------------------------------------------------

            else:
                print(cluster, 'UMAP Component 50, Best min cluster size is:␣
↪', hyper_param[umap_avg_metrics.index(max(umap_avg_metrics))][0], ' Best min␣
↪sample size is: ', hyper_param[umap_avg_metrics.
↪index(max(umap_avg_metrics))][1])
                print("UMAP Best Homogeneity: ", umap_homo[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                print("UMAP Best Completeness: ", umap_cm[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                print("UMAP Best V measure: ", umap_vm[umap_avg_metrics.
↪index(max(umap_avg_metrics))])
                print("UMAP Best Adjusted Rand Index: ",␣
↪umap_ar[umap_avg_metrics.index(max(umap_avg_metrics))])
                print("UMAP Best Adjusted Mutual Info Score: ",␣
↪umap_ami[umap_avg_metrics.index(max(umap_avg_metrics))])

                ␣
↪print("-------------------------------------------------------------------------------------


       if len(autoencoder_score) > 0:

           autoencoder_homo = autoencoder_score[:,0]
           autoencoder_cm = autoencoder_score[:,1]
           autoencoder_vm = autoencoder_score[:,2]
           autoencoder_ar = autoencoder_score[:,3]
           autoencoder_ami = autoencoder_score[:,4]

           for measure in zip(autoencoder_homo, autoencoder_cm,␣
↪autoencoder_vm, autoencoder_ar, autoencoder_ami):
             autoencoder_avg_metrics.append(sum(measure) / 5)

           if verbose == True:
             print(cluster, 'Autoencoder num_feature = 50, Best min cluster␣
↪size is: ', hyper_param[autoencoder_avg_metrics.
↪index(max(autoencoder_avg_metrics))][0], ' Best min sample size is: ',␣
↪hyper_param[autoencoder_avg_metrics.index(max(autoencoder_avg_metrics))][1])
             print("autoencoder Best Homogeneity: ",␣
↪autoencoder_homo[autoencoder_avg_metrics.
↪index(max(autoencoder_avg_metrics))])
```

```
            print("autoencoder Best Completeness: ",␣
↪autoencoder_cm[autoencoder_avg_metrics.index(max(autoencoder_avg_metrics))])
            print("autoencoder Best V measure: ",␣
↪autoencoder_vm[autoencoder_avg_metrics.index(max(autoencoder_avg_metrics))])
            print("autoencoder Best Adjusted Rand Index: ",␣
↪autoencoder_ar[autoencoder_avg_metrics.index(max(autoencoder_avg_metrics))])
            print("autoencoder Best Adjusted Mutual Info Score: ",␣
↪autoencoder_ami[autoencoder_avg_metrics.index(max(autoencoder_avg_metrics))])


          ␣
↪print("-----------------------------------------------------------------------------------------

   if withDimReduc == False and tune_db_hdb_best == False:

       wo_dr_score = np.array(wo_dr_score)
       wo_dr_avg_score = []

       homo = wo_dr_score[:,0]
       cm = wo_dr_score[:,1]
       vm = wo_dr_score[:,2]
       ar = wo_dr_score[:,3]
       ami = wo_dr_score[:,4]


       for measure in zip(homo, cm, vm, ar, ami):
           wo_dr_avg_score.append(sum(measure) / 5)

       if verbose == True:
         print(cluster,' without dimensionality reduction best component: ',␣
↪hyper_param[wo_dr_avg_score.index(max(wo_dr_avg_score))])
         print("Best Homogeneity: ", homo[wo_dr_avg_score.
↪index(max(wo_dr_avg_score))])
         print("Best Completeness: ", cm[wo_dr_avg_score.
↪index(max(wo_dr_avg_score))])
         print("Best V measure: ", vm[wo_dr_avg_score.
↪index(max(wo_dr_avg_score))])
         print("Best Adjusted Rand Index: ", ar[wo_dr_avg_score.
↪index(max(wo_dr_avg_score))])
         print("Best Adjusted Mutual Info Score: ", ami[wo_dr_avg_score.
↪index(max(wo_dr_avg_score))])
```

[ ]:
```
import hdbscan
from sklearn.cluster import DBSCAN

db_scores = []
hdb_scores = []
```

```
db_pred = DBSCAN(eps=0.5).fit_predict(umap_matrix_cos)
db_scores.append(report_five_scores(labels, db_pred))

clusterer = hdbscan.HDBSCAN(min_cluster_size=100, cluster_selection_epsilon = 0.
↪5)
clusterer.fit(umap_matrix_cos)
hdb_pred = clusterer.labels_
hdb_scores.append(report_five_scores(labels, hdb_pred))
```

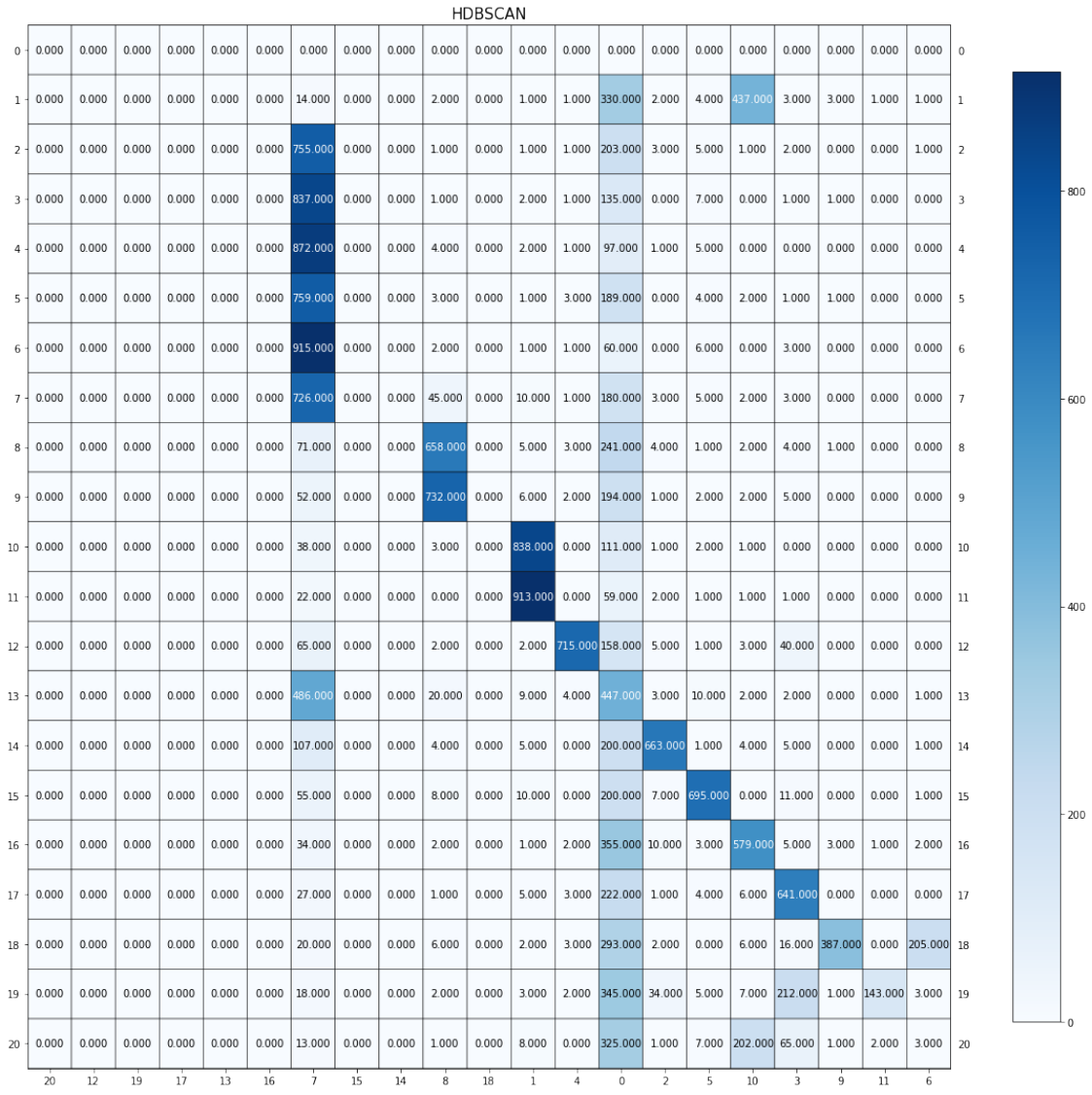
[ ]:
```
printResults(True, True, db_scores, hdb_scores, [], [], [], [], [], "", None,␣
↪None, [])
```

```
DBSCAN Homogeneity:  0.1516118290829085
DBSCAN Completeness:  0.5763459996381801
DBSCAN V-measure:  0.2400712451798938
DBSCAN Adjusted Random Index:  0.029695523163308122
DBSCAN Adjusted Mutual Information Score:  0.23412929886282383
----------------------------------------------------------------------
HDBSCAN Homogeneity:  0.4114112556788178
HDBSCAN Completeness:  0.607905615129666
HDBSCAN V-measure:  0.4907192642781056
HDBSCAN Adjusted Random Index:  0.19564202241842413
HDBSCAN Adjusted Mutual Information Score:  0.48958178555931836
```

Since TA Arash Vahab stated that we do not need to perform parameter tuning for Q15, we skipped the tuning part. We can see that from the 5 measures of both models, the performance of DBSCAN is far below the performance of HDBSCAN model. The reason might be because DBSCAN is prone to outliers/noises and can form false clusterings while HDBSCAN is focusing on high density data and that reduces the effects of outliers/noises.

1.0.16 QUESTION 16 Plot the contingency matrix for the best clustering model from Question 15. How many clusters are given by the model? What does "-1" mean for the clustering labels?

[ ]:
```
hdb_pred = hdbscan.HDBSCAN(min_cluster_size=100, cluster_selection_epsilon = 0.
↪5).fit_predict(umap_matrix_cos)
cm = confusion_matrix(labels, hdb_pred)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
↪title = 'HDBSCAN', size=(15,15))
```



How many clusters are given by the model? What does "-1" mean for the clustering labels? As we can observe from the contingency matrix, HDBSCAN has 12 clusters. "-1" means that in clustering there are outliers or noices that haven't been classified into any of the class labels.

1.0.17 QUESTION 17: Based on your experiments, which dimensionality reduction technique and clustering methods worked best together for 20-class text data and why? Follow the table below.

[ ]:
```
component_list = [5,20,200]
kmeans_list = [10,20,50]
scan_list = [100, 200]
eps_list = [0.5, 5]
```

```
svd_kscore = []
nmf_kscore = []
umap_kscore = []
hyper_param_kmean = []

#try K-means with SVD, NMF, and UMAP with all the given parameter values from␣
 ↪the table
for component in component_list:
    for k in kmeans_list:
        hyper_param_kmean.append([component, k])

        svd_matrix = TruncatedSVD(n_components=component).
 ↪fit_transform(total_tf_idf_matrix)
        nmf_matrix = NMF(n_components=component, tol=0.05, max_iter=50).
 ↪fit_transform(total_tf_idf_matrix)
        reducer_cos = umap.UMAP(n_components=component, metric="cosine")
        umap_matrix_cos = reducer_cos.fit_transform(total_tf_idf_matrix)

        kmeans_umap_cos = KMeans(n_clusters=k, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit(umap_matrix_cos)
        kmeans_svd = KMeans(n_clusters=k, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit(svd_matrix)
        kmeans_nmf = KMeans(n_clusters=k, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit(nmf_matrix)

        kmeans_svd_pred = kmeans_svd.predict(svd_matrix)
        kmeans_nmf_pred = kmeans_nmf.predict(nmf_matrix)
        kmeans_umap_pred = kmeans_umap_cos.predict(umap_matrix_cos)

        svd_kscore.append(report_five_scores(labels, kmeans_svd_pred))
        nmf_kscore.append(report_five_scores(labels, kmeans_nmf_pred))
        umap_kscore.append(report_five_scores(labels, kmeans_umap_pred))
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
```

```
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:1641: ConvergenceWarning: Maximum number
of iterations 50 reached. Increase it to improve convergence.
  ConvergenceWarning,
```

[ ]:
```
#try Agglomerative Clustering with SVD, NMF, and UMAP with all the given␣
↪parameter values from the table

svd_agscore = []
nmf_agscore = []
umap_agscore = []
hyper_param_agg = []

for component in component_list:
```

```
    hyper_param_agg.append([component, 20])

    svd_matrix = TruncatedSVD(n_components=component).
 ↪fit_transform(total_tf_idf_matrix)
    nmf_matrix = NMF(n_components=component, tol=0.05, max_iter=50).
 ↪fit_transform(total_tf_idf_matrix)
    reducer_cos = umap.UMAP(n_components=component, metric="cosine")
    umap_matrix_cos = reducer_cos.fit_transform(total_tf_idf_matrix)

    ward_pred_svd = AgglomerativeClustering(n_clusters=20, linkage='ward').
 ↪fit_predict(svd_matrix)
    ward_pred_nmf = AgglomerativeClustering(n_clusters=20, linkage='ward').
 ↪fit_predict(nmf_matrix)
    ward_pred_umap = AgglomerativeClustering(n_clusters=20, linkage='ward').
 ↪fit_predict(umap_matrix_cos)

    svd_agscore.append(report_five_scores(labels, ward_pred_svd))
    nmf_agscore.append(report_five_scores(labels, ward_pred_nmf))
    umap_agscore.append(report_five_scores(labels, ward_pred_umap))
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
```

[ ]:

```
#try DBSCAN with SVD, NMF, and UMAP with all the given parameter values from␣
 ↪the table

svd_dbscore = []
nmf_dbscore = []
umap_dbscore = []
hyper_param_dbscan = []

for component in component_list:
       for eps in eps_list:
              hyper_param_dbscan.append([component, eps])
```

```
            svd_matrix = TruncatedSVD(n_components=component).
↪fit_transform(total_tf_idf_matrix)
            nmf_matrix = NMF(n_components=component, tol=0.05, max_iter=50).
↪fit_transform(total_tf_idf_matrix)
            reducer_cos = umap.UMAP(n_components=component, metric="cosine")
            umap_matrix_cos = reducer_cos.fit_transform(total_tf_idf_matrix)

            db_pred_svd = DBSCAN(eps=eps, min_samples=60).
↪fit_predict(svd_matrix)
            db_pred_nmf = DBSCAN(eps=eps, min_samples=60).
↪fit_predict(nmf_matrix)
            db_pred_umap = DBSCAN(eps=eps, min_samples=60).
↪fit_predict(umap_matrix_cos)

            svd_dbscore.append(report_five_scores(labels, db_pred_svd))
            nmf_dbscore.append(report_five_scores(labels, db_pred_nmf))
            umap_dbscore.append(report_five_scores(labels, db_pred_umap))
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
```

```
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
```

[ ]:
```
#try HBDSCAN with SVD, NMF, and UMAP with all the given parameter values from␣
 ↪the table

svd_hdbscore = []
nmf_hdbscore = []
umap_hdbscore = []

hyper_param_hdbscan = []

for component in component_list:
    for cluster_size in scan_list:
        hyper_param_hdbscan.append([component, cluster_size])

        svd_matrix = TruncatedSVD(n_components=component).
 ↪fit_transform(total_tf_idf_matrix)
        nmf_matrix = NMF(n_components=component, tol=0.05, max_iter=50).
 ↪fit_transform(total_tf_idf_matrix)
        reducer_cos = umap.UMAP(n_components=component, metric="cosine")
        umap_matrix_cos = reducer_cos.fit_transform(total_tf_idf_matrix)

        clusterer = hdbscan.HDBSCAN(min_cluster_size=cluster_size,␣
 ↪min_samples=60)
        hdb_pred_svd = clusterer.fit_predict(svd_matrix)
        hdb_pred_nmf = clusterer.fit_predict(nmf_matrix)
        hdb_pred_umap = clusterer.fit_predict(umap_matrix_cos)

        svd_hdbscore.append(report_five_scores(labels, hdb_pred_svd))
        nmf_hdbscore.append(report_five_scores(labels, hdb_pred_nmf))
        umap_hdbscore.append(report_five_scores(labels, hdb_pred_umap))
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
```

```
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\decomposition\_nmf.py:294: FutureWarning: The 'init' value,
when 'init=None' and n_components is less than n_samples and n_features, will be
changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  FutureWarning,
```

[ ]:
```
#try K-means without dimensionality reduction

kmeans_wo_dr_score = []
for k in kmeans_list:
    kmeans_wo_dr_pred = KMeans(n_clusters=k, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit_predict(total_tf_idf_matrix)
    kmeans_wo_dr_score.append(report_five_scores(labels, kmeans_wo_dr_pred))
```

[ ]:
```
#try Agglomerative Clustering without dimensionality reduction

agg_wo_dr_score = []

ward_pred_wo_dr = AgglomerativeClustering(n_clusters=20, linkage='ward').
 ↪fit_predict(total_tf_idf_matrix.todense())
agg_wo_dr_score.append(report_five_scores(labels, ward_pred_wo_dr))
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\sklearn\utils\validation.py:598: FutureWarning: np.matrix usage is
deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy
array with np.asarray. For more information see:
https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
  FutureWarning,
```

[ ]:
```
#try DBSCAN without dimensionality reduction

db_wo_dr_score = []

for eps in eps_list:
    db_pred_wo_dr = DBSCAN(eps=eps, min_samples=60).
 ↪fit_predict(total_tf_idf_matrix)
    db_wo_dr_score.append(report_five_scores(labels, db_pred_wo_dr))
```

[ ]:
```
#try HBDSCAN without dimensionality reduction

hdb_wo_dr_score = []

for cluster_size in scan_list:
    hdb_pred_wo_dr = hdbscan.HDBSCAN(min_cluster_size=cluster_size,␣
 ↪min_samples=60).fit_predict(total_tf_idf_matrix)
    hdb_wo_dr_score.append(report_five_scores(labels, hdb_pred_wo_dr))
```

Below are the results from our experiments

[ ]:
```
print("5 measures of the Kmeans clustering with SVD, NMF, and UMAP dim␣
 ↪reduction")
print(" ")
printResults(True, False, [], [], svd_kscore, nmf_kscore, umap_kscore, [],␣
 ↪hyper_param_kmean, 'Kmeans Cluster', True, [], False)
```

```
5 measures of the Kmeans clustering with SVD, NMF, and UMAP dim reduction

SVD Component:  200 Kmeans Cluster :  50
SVD Best Homogeneity:  0.43610401645691743
SVD Best Completeness:  0.3916357466982703
SVD Best V measure:  0.41267540772038164
SVD Best Adjusted Rand Index:  0.12349601412269931
SVD Best Adjusted Mutual Info Score:  0.4079033326944282
--------------------------------------------------------------------------------
-------------------------
NMF Component:  20 Kmeans Cluster :  50
NMF Best Homogeneity:  0.37899726803809647
NMF Best Completeness:  0.32601304134602827
NMF Best V measure:  0.3505141708434653
NMF Best Adjusted Rand Index:  0.11717511222692681
NMF Best Adjusted Mutual Info Score:  0.34533092479893934
--------------------------------------------------------------------------------
-------------------------
UMAP Component:  5 Kmeans Cluster :  20
UMAP Best Homogeneity:  0.5712638714856522
UMAP Best Completeness:  0.5911372426780158
UMAP Best V measure:  0.5810306712834908
UMAP Best Adjusted Rand Index:  0.4564761953662068
UMAP Best Adjusted Mutual Info Score:  0.5796434571149668
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
print("5 measures of the Agglomerative clustering with SVD, NMF, and UMAP dim␣
 ↪reduction")
print(" ")
```

```
printResults(True, False, [], [], svd_agscore, nmf_agscore, umap_agscore, [],␣
 ↪hyper_param_agg, 'Agglomerative Cluster', True, [], False)
```

```
5 measures of the Agglomerative clustering with SVD, NMF, and UMAP dim reduction

SVD Component:  20 Agglomerative Cluster :  20
SVD Best Homogeneity:  0.37273689390300446
SVD Best Completeness:  0.4119060202368034
SVD Best V measure:  0.391343801865157
SVD Best Adjusted Rand Index:  0.18869877513646374
SVD Best Adjusted Mutual Info Score:  0.3892580652873262
--------------------------------------------------------------------------------
-------------------------
NMF Component:  20 Agglomerative Cluster :  20
NMF Best Homogeneity:  0.3618137897521295
NMF Best Completeness:  0.40205153968156543
NMF Best V measure:  0.38087287285498606
NMF Best Adjusted Rand Index:  0.15956692186808163
NMF Best Adjusted Mutual Info Score:  0.3787625728406377
--------------------------------------------------------------------------------
-------------------------
UMAP Component:  5 Agglomerative Cluster :  20
UMAP Best Homogeneity:  0.5607336129073461
UMAP Best Completeness:  0.587277509203352
UMAP Best V measure:  0.5736986918896229
UMAP Best Adjusted Rand Index:  0.4269142416887505
UMAP Best Adjusted Mutual Info Score:  0.5722883018870146
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
print("5 measures of the DBSCAN with SVD, NMF, and UMAP dim reduction")
print(" ")
printResults(True, False, [], [], svd_dbscore, nmf_dbscore, umap_dbscore, [],␣
 ↪hyper_param_dbscan, 'DBSCAN', True, [], False)
```

```
5 measures of the DBSCAN with SVD, NMF, and UMAP dim reduction

SVD Component:  5 DBSCAN :  0.5
SVD Best Homogeneity:  0.0
SVD Best Completeness:  1.0
SVD Best V measure:  0.0
SVD Best Adjusted Rand Index:  0.0
SVD Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
--------------------------------------------------------------------------------
-------------------------
NMF Component:  5 DBSCAN :  0.5
NMF Best Homogeneity:  0.0
```

```
NMF Best Completeness:  1.0
NMF Best V measure:  0.0
NMF Best Adjusted Rand Index:  0.0
NMF Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
--------------------------------------------------------------------------------
-------------------------
UMAP Component:  200 DBSCAN :  0.5
UMAP Best Homogeneity:  0.47055188098638845
UMAP Best Completeness:  0.5981452886767207
UMAP Best V measure:  0.5267317977059915
UMAP Best Adjusted Rand Index:  0.3066926396405913
UMAP Best Adjusted Mutual Info Score:  0.5255536899522504
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
print("5 measures of the DBSCAN with SVD, NMF, and UMAP dim reduction")
print(" ")
printResults(True, False, [], [], svd_hdbscore, nmf_hdbscore, umap_hdbscore,␣
 ↪[], hyper_param_hdbscan, 'HDBSCAN', True, [], False)
```

```
5 measures of the DBSCAN with SVD, NMF, and UMAP dim reduction

SVD Component:  5 HDBSCAN :  100
SVD Best Homogeneity:  0.0
SVD Best Completeness:  1.0
SVD Best V measure:  0.0
SVD Best Adjusted Rand Index:  0.0
SVD Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
--------------------------------------------------------------------------------
-------------------------
NMF Component:  20 HDBSCAN :  100
NMF Best Homogeneity:  0.0
NMF Best Completeness:  1.0
NMF Best V measure:  0.0
NMF Best Adjusted Rand Index:  0.0
NMF Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
--------------------------------------------------------------------------------
-------------------------
UMAP Component:  5 HDBSCAN :  100
UMAP Best Homogeneity:  0.4335166033379725
UMAP Best Completeness:  0.6324780626684263
UMAP Best V measure:  0.5144298562787586
UMAP Best Adjusted Rand Index:  0.22473153411281352
UMAP Best Adjusted Mutual Info Score:  0.5133524916010269
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
print("5 measures of the Kmeans clustering without dim reduction")
print(" ")
printResults(True, False, [], [], [], [], [], [], kmeans_list, 'Kmeans␣
 ↪Cluster', False, kmeans_wo_dr_score, False)
```

```
5 measures of the Kmeans clustering without dim reduction

Kmeans Cluster  without dimensionality reduction best component:  50
Best Homogeneity:  0.44122011132176103
Best Completeness:  0.38736577456218996
Best V measure:  0.4125427987280663
Best Adjusted Rand Index:  0.11161536082098349
Best Adjusted Mutual Info Score:  0.40786696496195946
```

[ ]:
```
print("5 measures of the Kmeans clustering without dim reduction")
print(" ")
printResults(True, False, [], [], [], [], [], [],[20], 'Agglomeratives␣
 ↪Cluster', False, agg_wo_dr_score, False)
```

```
5 measures of the Kmeans clustering without dim reduction

Agglomeratives Cluster  without dimensionality reduction best component:  20
Best Homogeneity:  0.3466740173500454
Best Completeness:  0.4628656696459231
Best V measure:  0.3964314628848338
Best Adjusted Rand Index:  0.11878829939415497
Best Adjusted Mutual Info Score:  0.39415477829650053
```

[ ]:
```
print("5 measures of the DBSCAN without dim reduction")
print(" ")
printResults(True, False, [], [], [], [], [], [], eps_list, 'DBSCAN', False,␣
 ↪db_wo_dr_score, False)
```

```
5 measures of the DBSCAN without dim reduction

DBSCAN  without dimensionality reduction best component:  0.5
Best Homogeneity:  0.0
Best Completeness:  1.0
Best V measure:  0.0
Best Adjusted Rand Index:  0.0
Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
```

[ ]:
```
print("5 measures of the HDBSCAN without dim reduction")
print(" ")
printResults(True, False, [], [], [], [], [], [], scan_list, 'HDBSCAN', False,␣
 ↪hdb_wo_dr_score, False)
```

```
5 measures of the HDBSCAN without dim reduction

HDBSCAN  without dimensionality reduction best component:  100
Best Homogeneity:  0.0
Best Completeness:  1.0
Best V measure:  0.0
Best Adjusted Rand Index:  0.0
Best Adjusted Mutual Info Score:  -1.6056889920887454e-16
```

From our experiments with different combinations, we observe that the best combination of clustering and dimensionality reduction techniques would be K-means clustering with k component equal to 20, and UMAP with n_component equal to 5. The five measures are relatively higher than the rest of the combinations. with homogeneity of 0.57, completeness of 0.59, v measure of 0.58, adjusted random index of 0.45, adjusted mutual information score of 0.58.

1.0.18 QUESTION 18: Bonus. If you can find creative ways to further enhance the clustering performance, report your method and the results you obtain.

[ ]:
```
from transformers import BertTokenizer, BertForSequenceClassification
import pandas as pd
import numpy as np
import torch

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',␣
 ↪num_labels=20, output_hidden_states=True).to(device)

# Load the fine tuned BERT model which has been saved locally
model.load_state_dict(torch.load("fine_tuned_bert.pth"))
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertForSequenceClassification: ['cls.seq_relationship.bias',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.predictions.bias', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
```

```
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:
```python
def bert_text_preparation(text, tokenizer):
    marked_text = "[CLS] " + text + " [SEP]"
    tokenized_text = tokenizer.tokenize(marked_text)
    # There exist long sentences in 20 news group,
    # so we constrain all token length to be less than 512
    tokenized_text = tokenized_text[:511]
    indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
    segments_ids = [1]*len(indexed_tokens)

    tokens_tensor = torch.tensor([indexed_tokens])
    segments_tensors = torch.tensor([segments_ids])

    return tokenized_text, tokens_tensor, segments_tensors

def get_bert_embeddings(tokens_tensor, segments_tensors, model):
    with torch.no_grad():
        tokens_tensor = tokens_tensor.to(device)
        segments_tensors = segments_tensors.to(device)
        outputs = model(tokens_tensor, segments_tensors)
        hidden_states = outputs.hidden_states

    token_embeddings = hidden_states[-1]
    token_embeddings = torch.squeeze(token_embeddings)
    # mean pooling of the last embedding layer
    token_embeddings = torch.mean(token_embeddings, dim=0).cpu()

    return token_embeddings
```

[ ]:
```python
bert_embeddings = []
for idx, text in enumerate(texts):
    tokenized_text, tokens_tensor, segments_tensors =␣
 ↪bert_text_preparation(text, tokenizer)
    emb = get_bert_embeddings(tokens_tensor, segments_tensors, model)
    bert_embeddings.append(emb)
embeddings = torch.stack(bert_embeddings).numpy()
print(embeddings.shape)
```

```
(18846, 768)
```

[ ]:
```python
reducer_cos = umap.UMAP(n_components=200, metric="cosine")
umap_matrix_cos = reducer_cos.fit_transform(embeddings)
```

```
kmeans_cos = KMeans(n_clusters=20, random_state=0, max_iter=2000, n_init=50).
 ↪fit(umap_matrix_cos)
kmeans_pred_cos = kmeans_cos.predict(umap_matrix_cos)

_ = report_five_scores(labels, kmeans_pred_cos, verbose=True)
```

```
homogeneity score is 0.8264822727819909
completeness score is 0.8383230653705785
v measure score is 0.8323605607390777
adjusted rand score is 0.7907152174652636
adjusted mutual info score is 0.8318114360018197
```

To improve clustering performance, our team chose to focus on improving the feature representation of texts. To do this, we use pretrained BERT and fine tuned it on text classification task with 20 news group dataset. Then, we use the last hidden layer of trained BERT as feature extractor and produce vectors of length 768 for every text sample. To offset the weakness of high dimensional clustering, we apply umap cosine dimension reduction on these 768-dimension vectors and reduce them to be 200-dimensional. Then, we simply apply kmeans on these vectors.

As shown above, all five metrics of BERT feature clustering outperform hugely compared to the results we obtained in Question 17.

*If you have doubts or concerns about the final results, please contact us and we will provide the source code of fine tuning BERT to graders.

1.0.19 Question 19 In a brief paragraph discuss: If the VGG network is trained on a dataset with perhaps totally different classes as targets, why would one expect the features derived from such a network to have discriminative power for a custom dataset?

Though VGG network is trained on different dataset, we can still assume that its feature extraction ability can still be transferred to another dataset. The reason is that as VGG falls in the paradigm of convolutional neural network, its kernels have been trained to capture certain features of an image (or more generally, patterns of pixels). Moreover, the ImageNet dataset is very large and contains a wide range of classes of objects. Therefore, it is safe to say that VGG has learnt to capture features of different objects and thus the features extracted by VGG have discriminative power. That being said, though we can directly apply VGG on a custom dataset and extract feature, a better way might be to train on the downstream dataset for a while before feature extraction.

1.0.20 Question 20 In a brief paragraph explain how the helper code base is performing feature extraction.

The helper code first downloads the flower dataset and VGG model online. Then it defines a extracter where it extracts the feature layers of VGG and applies average pooling and flattens the pooling output to a fully connected layer. Then, it creates a Pytorch dataloader and feeds flower image inputs into the defined extracter and gets the final feature representation.

1.0.21 Question 21 How many pixels are there in the original images? How many features does the VGG network extract per image; i.e what is the dimension of each feature vector for an image sample?

1. For each image, its raw dimension varies. For example, some have dimensions: (333, 500), (250, 320). After uniform operation of VGG, each image has three channels for each RGB color with size 224x224, which gives us 3x224x224=150528 pixels for each image.
2. The dimension of each feature vector is 4096.

1.0.22 Question 22 Are the extracted features dense or sparse? (Compare with sparse TF-IDF features in text.)

The extracted features are dense as there are rarely any 0's appearing.

1.0.23 Question 23 In order to inspect the high-dimensional features, t-SNE is a popular off-the-shelf choice for visualizing Vision features. Map the features you have extracted onto 2 dimensions with t-SNE. Then plot the mapped feature vectors along x and y axes. Color-code the data points with ground-truth labels. Describe your observation.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering
import torch
from torch import nn
import hdbscan
```

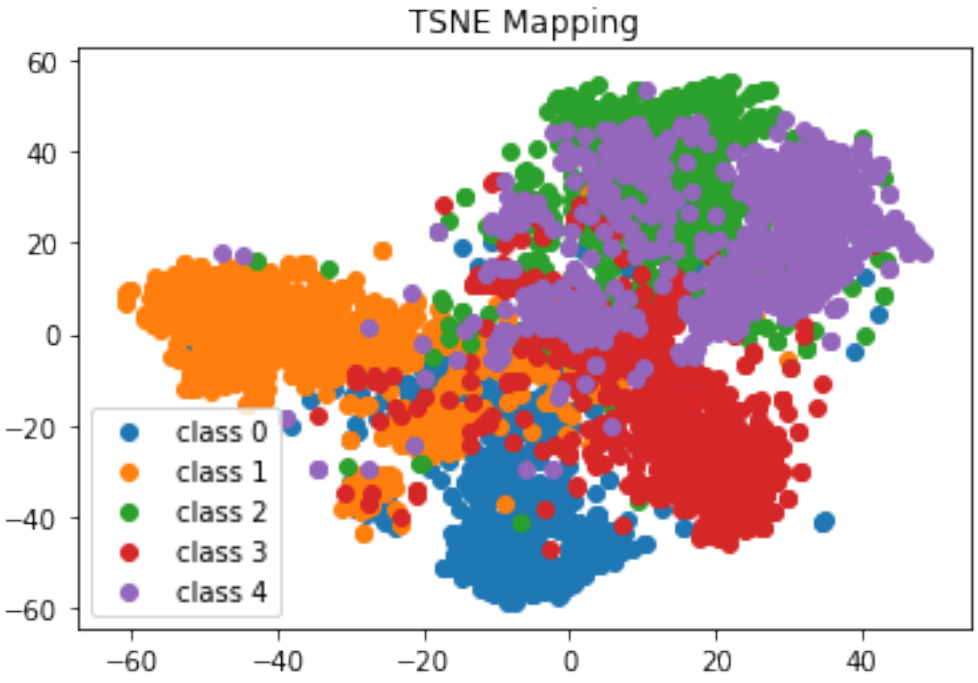
[ ]:
```
from sklearn.decomposition import TruncatedSVD
import umap.umap_ as umap
```

[ ]:
```
from sklearn import metrics

def report_five_scores(true_label, pred_label, verbose=False):
    homogeneity_score = metrics.homogeneity_score(true_label, pred_label)
    completeness_score = metrics.completeness_score(true_label, pred_label)
    v_measure_score = metrics.v_measure_score(true_label, pred_label)
    adjusted_rand_score = metrics.adjusted_rand_score(true_label, pred_label)
    adjusted_mutual_info_score = metrics.adjusted_mutual_info_score(true_label,␣
 ↪pred_label)

    if verbose:
        print("homogeneity score is {}".format(metrics.
 ↪homogeneity_score(true_label, pred_label)))
        print("completeness score is {}".format(metrics.
 ↪completeness_score(true_label, pred_label)))
        print("v measure score is {}".format(metrics.
 ↪v_measure_score(true_label, pred_label)))
        print("adjusted rand score is {}".format(metrics.
 ↪adjusted_rand_score(true_label, pred_label)))
```

```
        print("adjusted mutual info score is {}".format(metrics.
 ↪adjusted_mutual_info_score(true_label, pred_label)))

    return homogeneity_score, completeness_score, v_measure_score,␣
 ↪adjusted_rand_score, adjusted_mutual_info_score
```

[ ]:
```
from sklearn.manifold import TSNE
import os
filename = './flowers_features_and_labels.npz'

if os.path.exists(filename):
    file = np.load(filename)
    f_all, y_all = file['f_all'], file['y_all']
else:
    print("Run the helper code first before trying this problem!")

embedded = TSNE(n_components=2, learning_rate='auto', init='random').
 ↪fit_transform(f_all)
```

[ ]:
```
for i in range(5):
    plt.scatter(embedded[np.where(y_all==i)][:, 0], embedded[np.
 ↪where(y_all==i)][:, 1], label="class {}".format(i))
plt.legend()
plt.title("TSNE Mapping")
plt.show()
```



We can observe that there appears to be five clusters wich some outliers. Note that class4 and class 2 are distributed very closely which might be problematic for classification. Besides this, we can conclude that the feature extraction of VGG works fine on the flower dataset.

1.0.24 QUESTION 24: Report the best result (in terms of rand score) within the table below. For HDBSCAN introduce your own reasonable grid over min cluster size and min samples.

we do not need to do HDBSCAN without dimensionality reduction. see this post https://piazza.com/class/kxy3hn64g767gl?cid=128

[ ]:
```
import torch
from tqdm import tqdm
from torch import nn
from torch.utils.data import DataLoader, TensorDataset

class Autoencoder(torch.nn.Module):
    def __init__(self, n_components):
        super().__init__()
        self.n_components = n_components
        self.n_features = None  # to be determined with data
        self.encoder = None
        self.decoder = None

    def _create_encoder(self):
        return nn.Sequential(
            nn.Linear(4096, 1280),
            nn.ReLU(True),
            nn.Linear(1280, 640),
            nn.ReLU(True), nn.Linear(640, 120), nn.ReLU(True), nn.Linear(120,␣
 ↪self.n_components))

    def _create_decoder(self):
        return nn.Sequential(
            nn.Linear(self.n_components, 120),
            nn.ReLU(True),
            nn.Linear(120, 640),
            nn.ReLU(True),
            nn.Linear(640, 1280),
            nn.ReLU(True), nn.Linear(1280, 4096))

    def forward(self, X):
        encoded = self.encoder(X)
        decoded = self.decoder(encoded)
        return decoded

    def fit(self, X):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
```

```
        self.n_features = X.shape[1]
        self.encoder = self._create_encoder()
        self.decoder = self._create_decoder()
        self.cuda()
        self.train()

        criterion = nn.MSELoss()
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3,␣
↪weight_decay=1e-5)

        dataset = TensorDataset(X)
        dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

        for epoch in tqdm(range(100)):
            for (X_,) in dataloader:
                X_ = X_.cuda()
                # ===================forward====================
                output = self(X_)
                loss = criterion(output, X_)
                # ===================backward====================
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

        return self

    def transform(self, X):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
        self.eval()
        with torch.no_grad():
            return self.encoder(X).cpu().numpy()
```

[ ]:
```
#k means without dimensionality reduction
kmeans_dl_pred = KMeans(n_clusters=5, random_state=0, max_iter=1000, n_init=30).
↪fit_predict(f_all)
report_five_scores(y_all, kmeans_dl_pred, verbose=True)
```

```
homogeneity score is 0.32294205011802934
completeness score is 0.3572951348833885
v measure score is 0.3392511491596826
adjusted rand score is 0.18965701989591632
adjusted mutual info score is 0.33830208551751545
```

[ ]:
```
(0.32294205011802934,
 0.3572951348833885,
 0.3392511491596826,
 0.18965701989591632,
 0.33830208551751545)
```

[ ]:
```
#agglomerative clustering without dimensionality reduction
agg_dl_pred = AgglomerativeClustering(n_clusters=5).fit_predict(f_all)
report_five_scores(y_all, agg_dl_pred, verbose=True)
```

```
homogeneity score is 0.36884700410874754
completeness score is 0.41500566541515876
v measure score is 0.39056726430366534
adjusted rand score is 0.2184499487113686
adjusted mutual info score is 0.38968473891198474
```

[ ]:
```
(0.36884700410874754,
 0.41500566541515876,
 0.39056726430366534,
 0.2184499487113686,
 0.38968473891198474)
```

From Piazza post, TA specified that we can skip HDBSCAN without dimensionality reduction because it will take too long.

[ ]:
```
#k means with svd as dimensionality reduction technique
svd_matrix = TruncatedSVD(n_components=50).fit_transform(f_all)
kmeans_dl_svd_pred = KMeans(n_clusters=5, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit_predict(svd_matrix)
report_five_scores(y_all, kmeans_dl_svd_pred, verbose=True)
```

```
homogeneity score is 0.3244300493653036
completeness score is 0.35957423702245606
v measure score is 0.34109928779468063
adjusted rand score is 0.1891279845520144
adjusted mutual info score is 0.34015207468160125
```

[ ]:
```
(0.3244300493653036,
 0.35957423702245606,
 0.34109928779468063,
 0.1891279845520144,
 0.34015207468160125)
```

[ ]:
```
#k means with umap as dimensionality reduction technique
umap_metrix = umap.UMAP(n_components=50, metric="cosine").fit_transform(f_all)
kmeans_dl_umap_pred = KMeans(n_clusters=5, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit_predict(umap_metrix)
report_five_scores(y_all, kmeans_dl_umap_pred, verbose=True)
```

```
homogeneity score is 0.5331645187523326
completeness score is 0.5447644760285035
v measure score is 0.538902081865156
adjusted rand score is 0.4674467191613741
adjusted mutual info score is 0.5382651454516901
```

[ ]:
```
(0.5331645187523326,
 0.5447644760285035,
 0.538902081865156,
 0.4674467191613741,
 0.5382651454516901)
```

[ ]:
```
#k means with autoencoder
autoenc = Autoencoder(50).fit(f_all)
autoenc.fit(f_all)
ae_matrix = autoenc.transform(f_all)
kmeans_dl_ae_pred = KMeans(n_clusters=5, random_state=0, max_iter=1000,␣
 ↪n_init=30).fit_predict(ae_matrix)
report_five_scores(y_all, kmeans_dl_ae_pred, verbose=True)
```

```
100%|
| 100/100 [00:18<00:00,  5.35it/s]
100%|
| 100/100 [00:16<00:00,  6.01it/s]

homogeneity score is 0.3158310703800337
completeness score is 0.3417204580847193
v measure score is 0.32826610045189036
adjusted rand score is 0.22745459534999737
adjusted mutual info score is 0.3273114263413628
```

[ ]:
```
(0.3158310703800337,
 0.3417204580847193,
 0.32826610045189036,
 0.22745459534999737,
 0.3273114263413628)
```

[ ]:
```
#agglomerative clustering with svd
svd_matrix = TruncatedSVD(n_components=50).fit_transform(f_all)
agg_dl_svd_pred = AgglomerativeClustering(n_clusters=5).fit_predict(svd_matrix)
report_five_scores(y_all, agg_dl_svd_pred, verbose=True)
```

```
homogeneity score is 0.36433933624764675
completeness score is 0.40521696884076547
v measure score is 0.3836924744493291
adjusted rand score is 0.21765292937694478
adjusted mutual info score is 0.38280492874986766
```

[ ]:
```
(0.36433933624764675,
 0.40521696884076547,
 0.3836924744493291,
 0.21765292937694478,
 0.38280492874986766)
```

[ ]:
```
#agglomerative with umap
umap_metrix = umap.UMAP(n_components=50, metric="cosine").fit_transform(f_all)
agg_dl_umap_pred = AgglomerativeClustering(n_clusters=5).
 ↪fit_predict(umap_metrix)
report_five_scores(y_all, agg_dl_umap_pred, verbose=True)
```

```
homogeneity score is 0.5237980541467966
completeness score is 0.5506623966913001
v measure score is 0.5368943857425923
adjusted rand score is 0.4589545954937827
adjusted mutual info score is 0.5362455287618745
```

[ ]:
```
(0.5237980541467966,
 0.5506623966913001,
 0.5368943857425923,
 0.4589545954937827,
 0.5362455287618745)
```

[ ]:
```
#agglomerative with autoencoder
agg_dl_ae_pred = AgglomerativeClustering(n_clusters=5).fit_predict(ae_matrix)
report_five_scores(y_all, agg_dl_ae_pred, verbose=True)
```

```
homogeneity score is 0.2750220477783228
completeness score is 0.32500563926978876
v measure score is 0.297931973410131
adjusted rand score is 0.17197437717800568
adjusted mutual info score is 0.29689091286539054
```

[ ]:
```
(0.2750220477783228,
 0.32500563926978876,
 0.297931973410131,
 0.17197437717800568,
 0.29689091286539054)
```

[ ]:
```
min_cluster_size_list = [100, 110, 120, 130, 140, 150, 160, 170, 180]
min_sample_size_list = [10, 20, 30, 40, 50, 60]
hbd_score_svd = []
hbd_hyper_param = []

for min_cluster in min_cluster_size_list:
    for min_sample in min_sample_size_list:
        svd_matrix = TruncatedSVD(n_components=50).fit_transform(f_all)
        hbd_hyper_param.append([min_cluster, min_sample])
        hdb_dl_svd_pred = hdbscan.HDBSCAN(min_cluster_size=min_cluster,␣
 ↪min_samples=min_sample).fit_predict(svd_matrix)
        hbd_score_svd.append(report_five_scores(y_all, hdb_dl_svd_pred,␣
 ↪verbose=False))
```

[ ]:
```
hbd_score_umap = []

for min_cluster in min_cluster_size_list:
    for min_sample in min_sample_size_list:
        umap_metrix = umap.UMAP(n_components=50).fit_transform(f_all)
        hdb_dl_umap_pred = hdbscan.HDBSCAN(min_cluster_size=min_cluster,␣
 ↪min_samples=min_sample).fit_predict(umap_metrix)
        hbd_score_umap.append(report_five_scores(y_all, hdb_dl_umap_pred,␣
 ↪verbose=False))
```

[ ]:
```
hbd_score_ae = []

for min_cluster in min_cluster_size_list:
    for min_sample in min_sample_size_list:
        hdb_dl_ae_pred = hdbscan.HDBSCAN(min_cluster_size=min_cluster,␣
 ↪min_samples=min_sample).fit_predict(ae_matrix)
        hbd_score_ae.append(report_five_scores(y_all, hdb_dl_ae_pred,␣
 ↪verbose=False))
```

[ ]:
```
printResults(True, False, [], [], hbd_score_svd, [], [], [], hbd_hyper_param,␣
 ↪"HDBSCAN", True, [], True)
```

```
HDBSCAN SVD Component 50, Best min cluster size is:  100  Best min sample size
is:  10
SVD Best Homogeneity:  1.3876720518042915e-16
SVD Best Completeness:  1.0
SVD Best V measure:  2.7753441036085825e-16
SVD Best Adjusted Rand Index:  0.0
SVD Best Adjusted Mutual Info Score:  5.0062065300172974e-17
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
printResults(True, False, [], [], [], [], [], hbd_score_ae, hbd_hyper_param,␣
 ↪"HDBSCAN", True, [], True)
```

```
HDBSCAN Autoencoder num_feature = 50, Best min cluster size is:  100  Best min
sample size is:  10
autoencoder Best Homogeneity:  1.3876720518042915e-16
autoencoder Best Completeness:  1.0
autoencoder Best V measure:  2.7753441036085825e-16
autoencoder Best Adjusted Rand Index:  0.0
autoencoder Best Adjusted Mutual Info Score:  5.0062065300172974e-17
--------------------------------------------------------------------------------
-------------------------
```

[ ]:
```
printResults(True, False, [], [], [], [], hbd_score_umap, [], hbd_hyper_param,␣
 ↪"HDBSCAN", True, [], True)
```

```
# print(hbd_score_umap)
```

```
HDBSCAN UMAP Component 50, Best min cluster size is:  120  Best min sample size
is:  10
UMAP Best Homogeneity:  0.5096713103325441
UMAP Best Completeness:  0.4609138611731552
UMAP Best V measure:  0.4840679179347527
UMAP Best Adjusted Rand Index:  0.38947062955837736
UMAP Best Adjusted Mutual Info Score:  0.483060957696171
--------------------------------------------------------------------------------
-------------------------
```

From our experiments, we can observe that the best result in terms of random score would be the combination of UMAP as dim reduction and Kmeans as clustering technique. The adjusted rand score we got is 0.4674467191613741, which is the highest among all the other combinations.

1.0.25 Question 25 Report the test accuracy of the MLP classifier on the original VGG features.Report the same when using the reduced-dimension features (you have freedom in choosing the dimensionality reduction algorithm and its parameters). Does the performance of the model suffer with the reduced-dimension representations? Is it significant? Correlate your classification results with the clustering results obtained for the same features in Question 24.

[ ]:
```
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, TensorDataset
from tqdm import tqdm

class MLP(torch.nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(num_features, 1280),
            nn.ReLU(True),
            nn.Linear(1280, 640),
            nn.ReLU(True),
            nn.Linear(640, 5),
            nn.LogSoftmax(dim=1)
        )
        self.cuda()


    def forward(self, X):
        return self.model(X)

    def train(self, X, y):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
```

```
        y = torch.tensor(y, dtype=torch.int64, device='cuda')

        self.model.train()

        criterion = nn.NLLLoss()
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3,␣
↪weight_decay=1e-5)

        dataset = TensorDataset(X, y)
        dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

        for epoch in tqdm(range(100)):
            for (X_, y_) in dataloader:
                optimizer.zero_grad()
                output = self.forward(X_)
                loss = criterion(output, y_)

                loss.backward()
                optimizer.step()

        return self

    def eval(self, X_test, y_test):
        X_test = torch.tensor(X_test, dtype=torch.float32, device='cuda')
        y_test = torch.tensor(y_test, dtype=torch.int64, device='cuda')

        accuracy = None
        match_count, total_count = 0, 0
        dataset = TensorDataset(X_test, y_test)
        dataloader = DataLoader(dataset, batch_size=128)
        for (X, y) in dataloader:
            soft_probs = self.forward(X)
            match_count += torch.sum(torch.argmax(soft_probs, dim=1) == y)
            total_count += len(X)
        return match_count / total_count
```

```
c:\users\yivyu\appdata\local\programs\python\python37\lib\site-
packages\torchvision\io\image.py:11: UserWarning: Failed to load image Python
extension: [WinError 126] The specified module could not be found
  warn(f"Failed to load image Python extension: {e}")
```

[ ]:
```
from sklearn.model_selection import train_test_split
import random

np.random.seed(42)
random.seed(42)
```

```
train_data, test_data, train_label, test_label = train_test_split(f_all, y_all,␣
↪test_size=0.2)
```

[ ]:
```
mlp = MLP(4096)
_ = mlp.train(train_data, train_label)
```

```
100%|
| 100/100 [00:06<00:00, 16.17it/s]
```

[ ]:
```
print(mlp.eval(test_data, test_label))
```

```
tensor(0.9142, device='cuda:0')
```

[ ]:
```
svd = TruncatedSVD(n_components=100).fit(train_data)
reduced_train_data = svd.transform(train_data)
reduced_test_data = svd.transform(test_data)
```

[ ]:
```
mlp_reduced = MLP(100)
_ = mlp_reduced.train(reduced_train_data, train_label)
```

```
100%|
| 100/100 [00:05<00:00, 19.27it/s]
```

[ ]:
```
print(mlp_reduced.eval(reduced_test_data, test_label))
```

```
tensor(0.9019, device='cuda:0')
```

As shown above, the test accuracy of MLP classifier on the original VGG features is 0.91 while the reduced-dim features on the MLP classifier have 0.9019 accuracy. The performance does suffer with the dimension reduction, but not significantly. Compared to the result of question 24, where the reduced-dimension feature has the best performance, we can see there is a discrepency. We reason that this might be because for clustering, especially for kmeans, if the feature space is high dimensional then clustering algorithms will have trouble distinguishing the distance between vectors due to curse of dimensionality. On the other hand, for a neural network, high dimensional feature space is not a big problem and some times even bigger dimension carries more information.

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('project2_final.ipynb')
```

```
--2022-02-07 07:06:05--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2022-02-07 07:06:05 (14.3 MB/s) - colab_pdf.py saved [1864/1864]

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```