Preparing data

In [1]:
```
from plotmat import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
np.random.seed(0)
random.seed(0)
from sklearn.datasets import fetch_20newsgroups
newsgroups_train = fetch_20newsgroups(subset='train')
print(newsgroups_train.keys())
#print(newsgroups_train.data[1])
#print(newsgroups_train.data[0])
print(newsgroups_train.target)
print((newsgroups_train.target_names))
```

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
[7 4 4 ... 3 1 8]
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardwar
e', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey',
'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.p
olitics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

Question 1

In [2]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer


categories = ['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']

data = fetch_20newsgroups(subset='all', categories=categories, shuffle=True, random_state=0,remove=('headers','
vectorizer = CountVectorizer(stop_words='english', min_df=3)
#vectorizer = CountVectorizer(min_df=3)
data_vec = vectorizer.fit_transform(data.data)
tfidf = TfidfTransformer()
data_tfidf = tfidf.fit_transform(data_vec)
print("train count size ", data_tfidf.shape)
```

```
train count size  (7882, 23522)
```

Q1: Report the dimensions of the TF-IDF matrix you obtain.

The shape is (7882, 23522)

Question 2

In [3]:
```
print(data.target_names)
print(len(data.target))
```

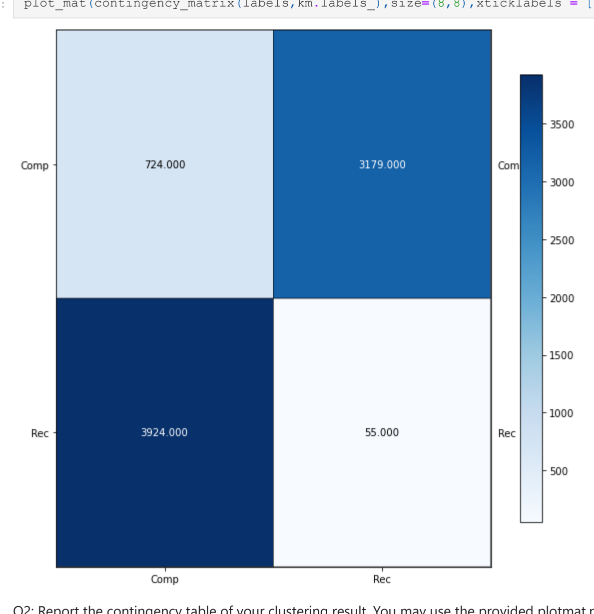
```
['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'rec.autos',
'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey']
7882
```

In [4]:
```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import contingency_matrix

labels = []
comp = ['comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.pc.hardware','comp.sys.mac.hardware']
rec = ['rec.autos','rec.motorcycles','rec.sport.baseball','rec.sport.hockey']
for target in data.target:
    name = data.target_names[target]
    #print(name)
    if name in comp:
        labels.append(0)
    elif name in rec:
        labels.append(1)

#random state=0, max iter ≥ 1000 and n init ≥ 30
km = KMeans(n_clusters=2, max_iter=1000, n_init=30, random_state=0)
km.fit(data_tfidf)
```

Out[4]:
```
KMeans(max_iter=1000, n_clusters=2, n_init=30, random_state=0)
```

In [5]:
```
#labels
```

In [6]:
```
plot_mat(contingency_matrix(labels,km.labels_),size=(8,8),xticklabels = ['Comp','Rec'],yticklabels = ['Comp','R
```



Q2: Report the contingency table of your clustering result. You may use the provided plotmat.py to visualize the matrix. Does the contingency matrix have to be square-shaped?

The plot is shown above. The shape has not to be square-shaped. It may be flat or tall. The numbers of class and cluster generated in row and column might not be the same, specifically the length might be different. Sometimes you may come up with many clusters, comparing with a few classes.

Question 3

In [7]:
```
from sklearn.metrics.cluster import homogeneity_score, completeness_score, v_measure_score, adjusted_rand_score
```

In [8]:
```
print("The homogeneity score is:", homogeneity_score(labels, km.labels_))
print("The completeness_score score is:", completeness_score(labels, km.labels_))
print("The V-measure score is:", v_measure_score(labels, km.labels_))
print("The adjusted Rand Index score is:", adjusted_rand_score(labels, km.labels_))
print("The adjusted mutual information score is:", adjusted_mutual_info_score(labels, km.labels_))
```

```
The homogeneity score is: 0.5809557696057134
The completeness_score score is: 0.5948001867166814
The V-measure score is: 0.5877964697988065
The adjusted Rand Index score is: 0.6436957246204892
The adjusted mutual information score is: 0.587758287733538
```

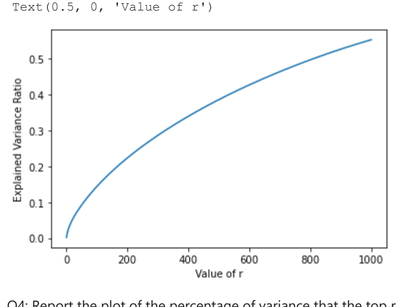
Q3: Report the 5 clustering measures explained in the introduction for Kmeans clustering.

Shown above

Question 4

In [9]:
```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=1000, random_state=0)
LSI = svd.fit_transform(data_tfidf)

x = np.array(range(1, 1001))
ratio = svd.explained_variance_ratio_.cumsum()
plt.plot(x, ratio)
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Value of r')
```

Out[9]:
```
Text(0.5, 0, 'Value of r')
```



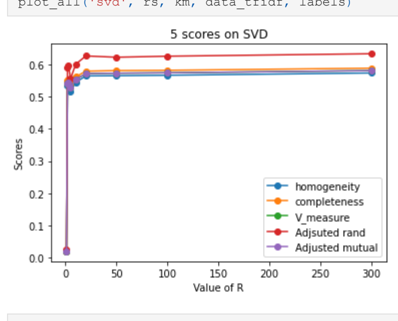
Q4: Report the plot of the percentage of variance that the top r principle components retain v.s. r, for r = 1 to 1000.

Shown above.

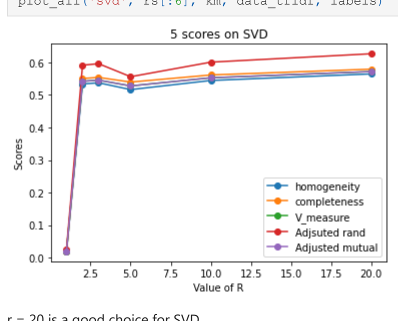
Question 5

In [10]:
```
from sklearn.decomposition import NMF
import umap.umap_ as umap

def plot_all(reducer, rs, km, data_tfidf, labels):
    hs,cs,vs,aris,amis = [],[],[],[],[]
    for r in rs:
        if reducer == 'svd':
            svd = TruncatedSVD(n_components=r, random_state=0)
            data_r = svd.fit_transform(data_tfidf)
        elif reducer == 'nmf':
            # max_iter = 500
            nmf = NMF(n_components=r, max_iter = 500, random_state=0)
            data_r = nmf.fit_transform(data_tfidf)
        elif reducer == 'euc':
            euc = umap.UMAP(n_components=r, metric='euclidean', random_state=0)
            data_r = euc.fit_transform(data_tfidf)
        elif reducer == 'cos':
            cos = umap.UMAP(n_components=r, metric='cosine', random_state=0)
            data_r = cos.fit_transform(data_tfidf)

        km_fit = km.fit(data_r)
        hs.append(homogeneity_score(labels,km_fit.labels_))
        cs.append(completeness_score(labels,km_fit.labels_))
        vs.append(v_measure_score(labels,km_fit.labels_))
        aris.append(adjusted_rand_score(labels,km_fit.labels_))
        amis.append(adjusted_mutual_info_score(labels,km_fit.labels_))

    plt.plot(rs, hs, marker="o")
    plt.plot(rs, cs, marker="o")
    plt.plot(rs, vs, marker="o")
    plt.plot(rs, aris, marker="o")
    plt.plot(rs, amis, marker="o")
    plt.title('5 scores on {}'.format(reducer.upper()))
    plt.xlabel('Value of R')
    plt.ylabel('Scores')
    plt.legend(labels = ['homogeneity','completeness', 'V_measure','Adjsuted rand','Adjusted mutual'])
```

In [11]:
```
rs = [1, 2, 3, 5, 10, 20, 50, 100, 300]
plot_all('svd', rs, km, data_tfidf, labels)
```



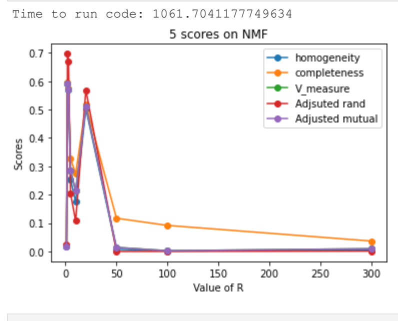
In [12]:
```
plot_all('svd', rs[:6], km, data_tfidf, labels)
```



r = 20 is a good choice for SVD

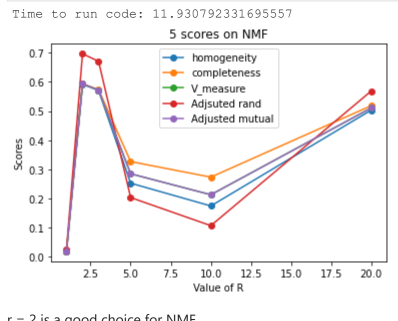
In [13]:
```
import time

time_start =time.time()
plot_all('nmf', rs, km, data_tfidf, labels)
print('Time to run code: {}'.format(time.time()-time_start))
```

```
Time to run code: 1061.7041177749634
```



In [14]:
```
time_start =time.time()
plot_all('nmf', rs[:6], km, data_tfidf, labels)
print('Time to run code: {}'.format(time.time()-time_start))
```

```
Time to run code: 11.930792331695557
```



r = 2 is a good choice for NMF

Q5 Report a good choice of r for SVD and NMF respectively.

From the aboved plots, r = 20 is a good choice for SVD r = 2 is a good choice for NMF

Question 6

Q6: How do you explain the non-monotonic behavior of the measures as r increases?

When used for topic modeling in natural language processing, n_components (r) can be used as the number of topics to print out. They will try to retain the principal components, namely features, while reducing the dimension of features to some extent. However, in this case, the relation between dimension r and K-means results are non-monotonic. When r is too small, the vector can only keep little information(features) of original data, so the results won't be good; On the other hand when r is too large, the noisy feature matrix will degrade K-means clustering performance, and the data are in a high-dimensional space so that the Euclidean distance is not a good metric. The linear dimension reduction effect like PCA and NMF will be relatively simple in this case because the data itself may be embedded in the manifold in a high-dimensional space.

Question 7

In [15]:
```
rs = [1, 2, 3, 5, 10, 20, 50, 100, 300]

import statistics
def compute_average(reducer, rs, km, data_tfidf, labels):
    hs,cs,vs,aris,amis = [],[],[],[],[]
    for r in rs:
        if reducer == 'svd':
            svd = TruncatedSVD(n_components=r, random_state=0)
            data_r = svd.fit_transform(data_tfidf)
        elif reducer == 'nmf':
            nmf = NMF(n_components=r, max_iter = 500, random_state=0)
            data_r = nmf.fit_transform(data_tfidf)

        km.fit(data_r)
        hs.append(homogeneity_score(labels,km.labels_))
        cs.append(completeness_score(labels,km.labels_))
        vs.append(v_measure_score(labels,km.labels_))
        aris.append(adjusted_rand_score(labels,km.labels_))
        amis.append(adjusted_mutual_info_score(labels,km.labels_))

    print("The mean homogeneity score of {} is: {}".format(reducer.upper(), statistics.mean(hs)))
    print("The mean completeness_score of {} score is: {}".format(reducer.upper(), statistics.mean(cs)))
    print("The mean V-measure score of {} is: {}".format(reducer.upper(), statistics.mean(vs)))
    print("The mean adjusted Rand Index score of {} is: {}".format(reducer.upper(), statistics.mean(aris)))
    print("The mean adjusted mutual information score of {} is: {}".format(reducer.upper(), statistics.mean(ami
    #('5 scores on {}'.format('SVD'))
```

In [16]:
```
time_start =time.time()

#SVD
compute_average('svd', rs, km, data_tfidf, labels)
print('\n')
#NMF
compute_average('nmf', rs, km, data_tfidf, labels)
print('\n')
print('Time to run code: {}'.format(time.time()-time_start))
```

```
The mean homogeneity score of SVD is: 0.49130160645561705
The mean completeness_score of SVD score is: 0.5063017751099483
The mean V-measure score of SVD is: 0.4986851154350201
The mean adjusted Rand Index score of SVD is: 0.5421750392980184
The mean adjusted mutual information score of SVD is: 0.49863858271416794


The mean homogeneity score of NMF is: 0.23640561724554449
The mean completeness_score of NMF score is: 0.28351592127322395
The mean V-measure score of NMF is: 0.24670292234205082
The mean adjusted Rand Index score of NMF is: 0.2527153641303994
The mean adjusted mutual information score of NMF is: 0.24660352778375802


Time to run code: 1078.2093074321747
```

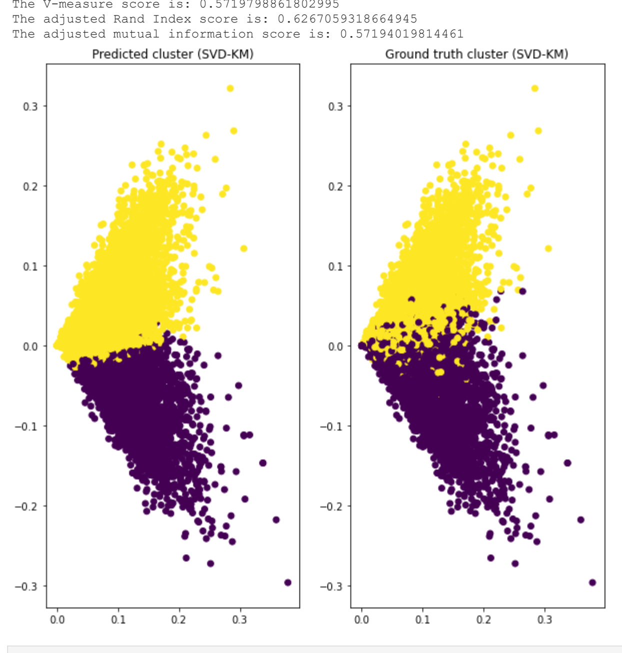
Q7: Are these measures on average better than those computed in Question 3?

They are not. r=1 for both models lower the arerage score.

Question 8

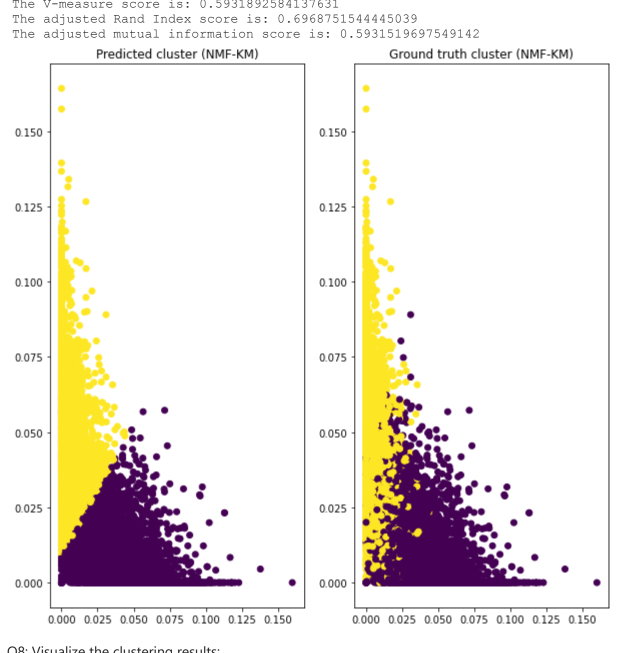
In [17]:
```
# SVD r = 20
svd_best = TruncatedSVD(n_components=20, random_state=0)
data_svd_best = svd_best.fit_transform(data_tfidf)

plt.figure(figsize=(10, 10))
km.fit(data_svd_best)
plt.subplot(121)
plt.scatter(data_svd_best[:, 0], data_svd_best[:, 1], c=km.labels_)
plt.title("Predicted cluster (SVD-KM)")

plt.subplot(122)
plt.scatter(data_svd_best[:, 0], data_svd_best[:, 1], c=labels)
plt.title("Ground truth cluster (SVD-KM)")

print("The homogeneity score is:", homogeneity_score(labels, km.labels_))
print("The completeness_score score is:", completeness_score(labels, km.labels_))
print("The V-measure score is:", v_measure_score(labels, km.labels_))
print("The adjusted Rand Index score is:", adjusted_rand_score(labels, km.labels_))
print("The adjusted mutual information score is:", adjusted_mutual_info_score(labels, km.labels_))
```

```
The homogeneity score is: 0.5647412437625056
The completeness_score score is: 0.5794065024081754
The V-measure score is: 0.5719798861802995
The adjusted Rand Index score is: 0.6267059318664945
The adjusted mutual information score is: 0.57194019814461
```



In [18]:
```
# NMF r = 2
nmf_best = NMF(n_components=2, random_state=0)
data_nmf_best = nmf_best.fit_transform(data_tfidf)

plt.figure(figsize=(10, 10))
km.fit(data_nmf_best)
plt.subplot(121)
plt.scatter(data_nmf_best[:, 0], data_nmf_best[:, 1], c=km.labels_)
plt.title("Predicted cluster (NMF-KM)")

plt.subplot(122)
plt.scatter(data_nmf_best[:, 0], data_nmf_best[:, 1], c=labels)
plt.title("Ground truth cluster (NMF-KM)")

print("The homogeneity score is:", homogeneity_score(labels, km.labels_))
print("The completeness_score score is:", completeness_score(labels, km.labels_))
print("The V-measure score is:", v_measure_score(labels, km.labels_))
print("The adjusted Rand Index score is:", adjusted_rand_score(labels, km.labels_))
print("The adjusted mutual information score is:", adjusted_mutual_info_score(labels, km.labels_))
```

```
The homogeneity score is: 0.5924736270587376
The completeness_score score is: 0.5939066206393134
The V-measure score is: 0.5931892584137631
The adjusted Rand Index score is: 0.6968751544445039
The adjusted mutual information score is: 0.5931519697549142
```



Q8: Visualize the clustering results:

They are shown above

Question 9

Q9: 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 are projecting the data vectors onto 2D dimensional plot. For both SVD and NMF, we got a simliar plot of predicted cluster and ground-truth cluster. Two clusters in both cases are in up and down but not in spherical. They are close together and some boundaries are overlapping with each other. This is beacuse the centroids are closed to each other. Besides we can see that there are some outliers and noisy vertors cases for the predicted clusters. Certainly, KM clustering is not ideal for both cases.

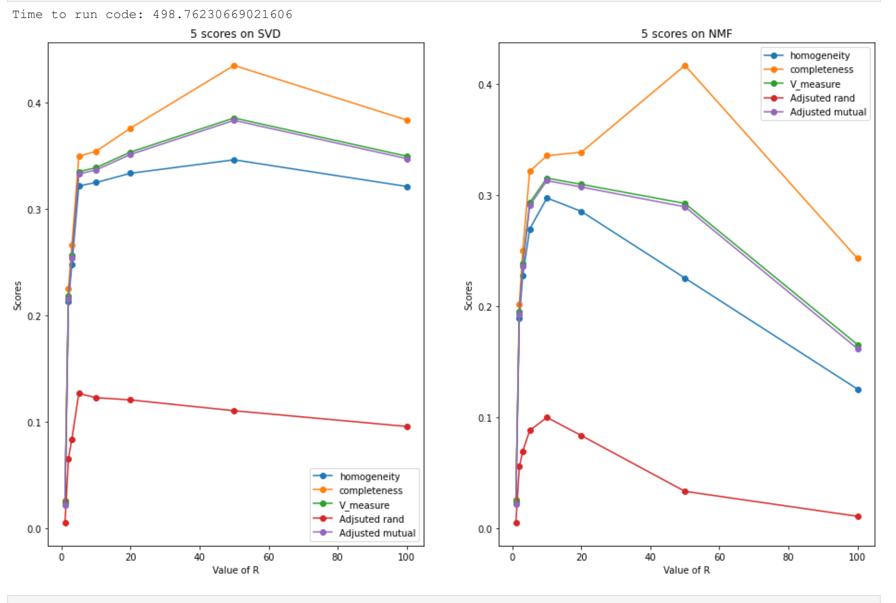
Question 10

In [19]:
```
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
              'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
              'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
              'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
              'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
              'talk.politics.misc', 'talk.religion.misc']

data_all = fetch_20newsgroups(subset='all', categories=categories, shuffle=True, random_state=0,remove=('header
vectorizer = CountVectorizer(stop_words='english', min_df=3)
tfidf = TfidfTransformer()
data_all_vec = vectorizer.fit_transform(data_all.data)
data_all_tfidf = tfidf.fit_transform(data_all_vec)
print("train count size ", data_all_tfidf.shape)
```

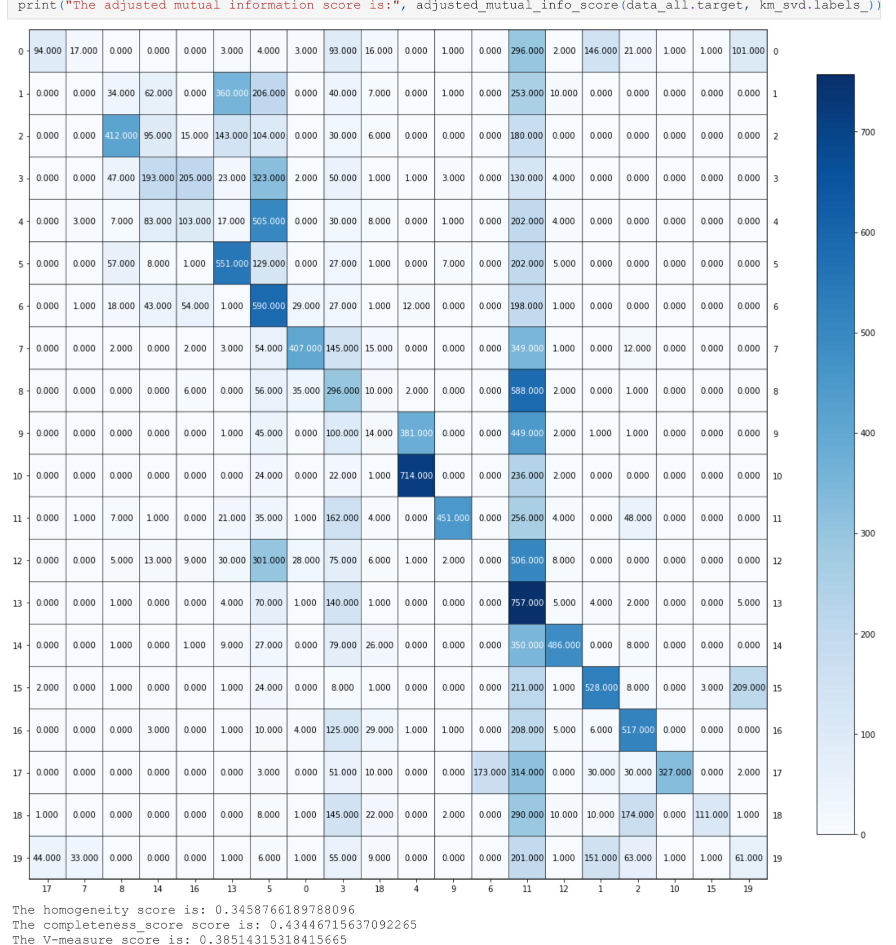
```
train count size  (18846, 45365)
```

In [20]:
```
# Finding the best parameter from SVD and NMF
# The list of r has been shorten after testing from the orginal candiates in order to save time and memory.

time_start =time.time()
rs = [1, 2, 3, 5, 10, 20, 50, 100]
km = KMeans(n_clusters=20, max_iter=1000, n_init=30, random_state=0)
plt.figure(figsize=(16, 10))
plt.subplot(121)
plot_all('svd', rs, km, data_all_tfidf, data_all.target)
plt.subplot(122)
plot_all('nmf', rs, km, data_all_tfidf, data_all.target)

print('Time to run code: {}'.format(time.time()-time_start))
```

```
Time to run code: 498.76230669021606
```



In [21]:
```
# Best r for SVD is 50
# Best r for NMF is 50
# I will manually choose SVD = 50 here since it scores better.

km = KMeans(n_clusters=20, max_iter=1000, n_init=30, random_state=0)
svd_Q10 = TruncatedSVD(n_components=50, random_state=0)
data_Q10_r = svd_Q10.fit_transform(data_all_tfidf)
km_svd = km.fit(data_Q10_r)

cm = confusion_matrix(data_all.target, km_svd.labels_)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], size=(15,15), xticklabels=cols, yticklabels=rows)

print("The homogeneity score is:", homogeneity_score(data_all.target, km_svd.labels_))
print("The completeness_score score is:", completeness_score(data_all.target, km_svd.labels_))
print("The V-measure score is:", v_measure_score(data_all.target, km_svd.labels_))
print("The adjusted Rand Index score is:", adjusted_rand_score(data_all.target, km_svd.labels_))
print("The adjusted mutual information score is:", adjusted_mutual_info_score(data_all.target, km_svd.labels_))
```



```
The homogeneity score is: 0.3458766189788096
The completeness_score score is: 0.43446715637092265
The V-measure score is: 0.38514315318415665
The adjusted Rand Index score is: 0.11042604980495854
The adjusted mutual information score is: 0.38290635686500557
```

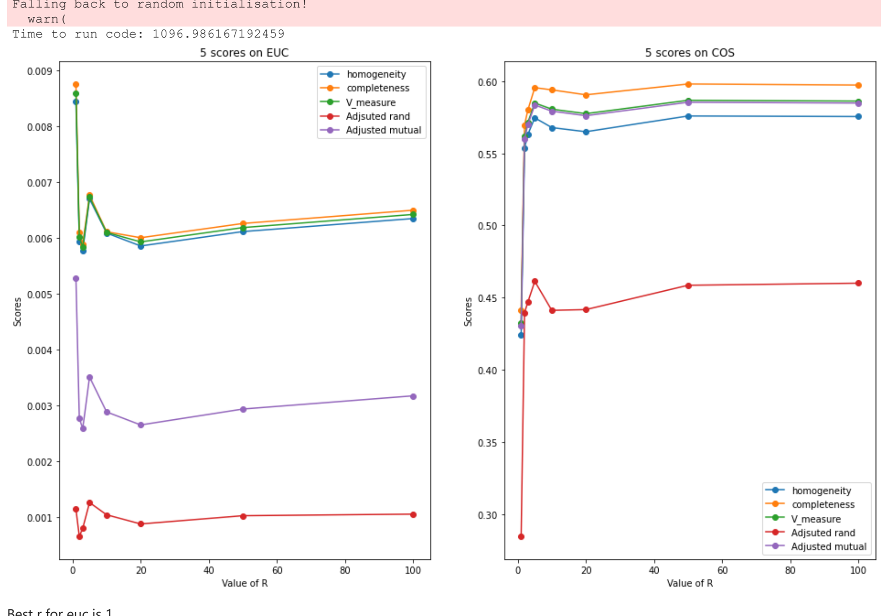
Q10: Load documents with the same configuration as in Question 1, but for ALL 20 categories.

Results are shown above.

Question 11

In [22]:
```
# The list of r has been shorten after testing from the orginal candiates in order to save time and memory.

import umap.umap_ as umap
time_start =time.time()
rs = [1, 2, 3, 5, 10, 20, 50, 100]
km = KMeans(n_clusters=20, max_iter=1000, n_init=30, random_state=0)
plt.figure(figsize=(16, 10))
plt.subplot(121)
plot_all('euc', rs, km, data_all_tfidf, data_all.target)
plt.subplot(122)
plot_all('cos', rs, km, data_all_tfidf, data_all.target)

print('Time to run code: {}'.format(time.time()-time_start))
```

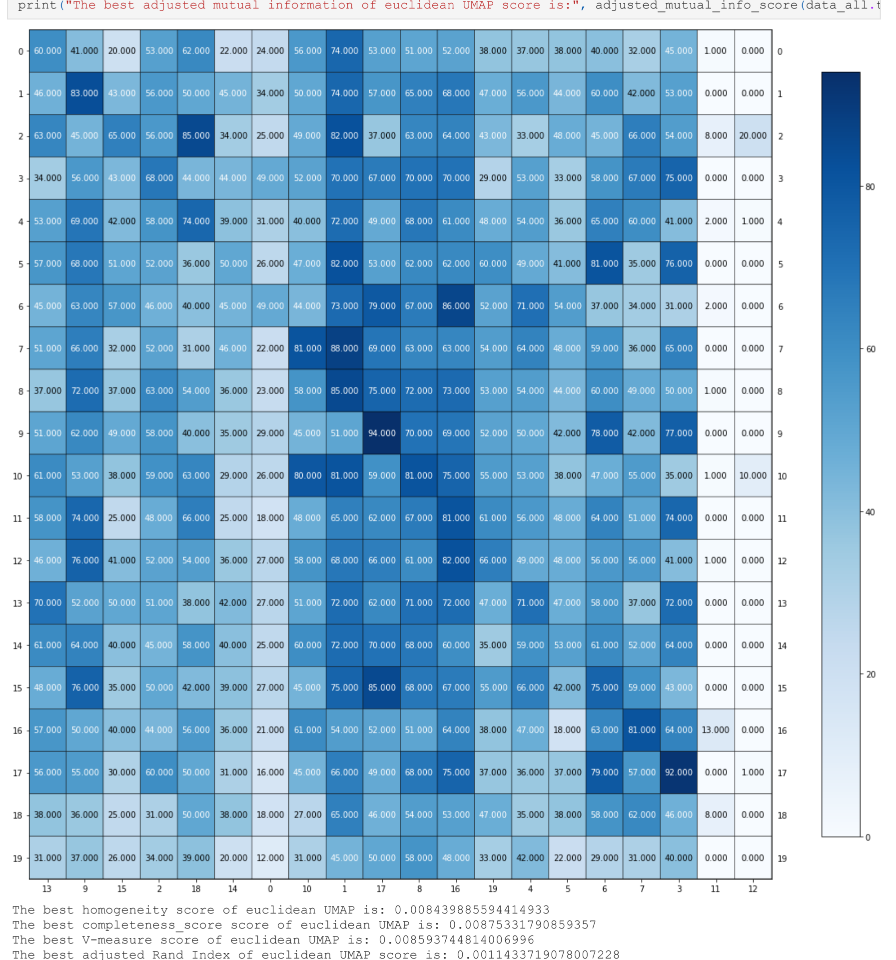
```
C:\Users\ericz\anaconda3\lib\site-packages\umap\spectral.py:260: UserWarning: WARNING: spectral initialisation
failed! The eigenvector solver
failed. This is likely due to too small an eigengap. Consider
adding some noise or jitter to your data.

Falling back to random initialisation!
  warn(
Time to run code: 1096.986167192459
```



Best r for euc is 1.

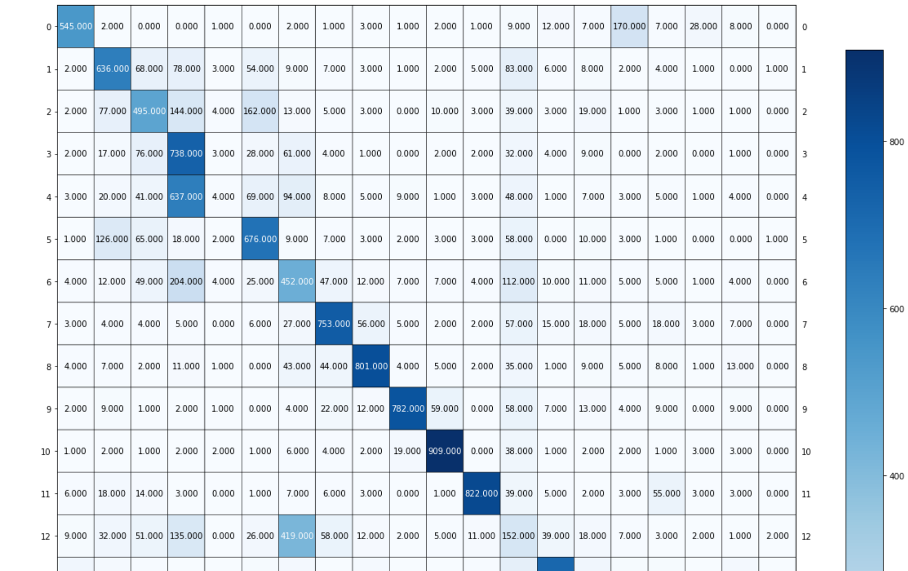
Best r for cos is 5.

In [23]:
```
km = KMeans(n_clusters=20, max_iter=1000, n_init=30, random_state=0)

euc_Q11 = umap.UMAP(n_components=1, metric='euclidean', random_state=0)
data_Q11_euc_r = euc_Q11.fit_transform(data_all_tfidf)
km_euc = km.fit(data_Q11_euc_r)
cm = confusion_matrix(data_all.target, km_euc.labels_)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], size=(15,15), xticklabels=cols, yticklabels=rows)

print("The best homogeneity score of euclidean UMAP is:", homogeneity_score(data_all.target, km_euc.labels_))
print("The best completeness_score score of euclidean UMAP is:", completeness_score(data_all.target, km_euc.lab
print("The best V-measure score of euclidean UMAP is:", v_measure_score(data_all.target, km_euc.labels_))
print("The best adjusted Rand Index of euclidean UMAP score is:", adjusted_rand_score(data_all.target, km_euc.l
print("The best adjusted mutual information of euclidean UMAP score is:", adjusted_mutual_info_score(data_all.t
```



```
The best homogeneity score of euclidean UMAP is: 0.008439885594414933
The best completeness_score score of euclidean UMAP is: 0.00875331790859357
The best V-measure score of euclidean UMAP is: 0.008593744814006996
The best adjusted Rand Index of euclidean UMAP score is: 0.0011433719078007228
The best adjusted mutual information of euclidean UMAP score is: 0.005283855445413727
```

In [24]:
```
#r = 5

cos_Q11 = umap.UMAP(n_components=5, metric='cosine', random_state=0)
data_Q11_cos_r = cos_Q11.fit_transform(data_all_tfidf)
km_cos = km.fit(data_Q11_cos_r)
cm = confusion_matrix(data_all.target, km_cos.labels_)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], size=(15,15), xticklabels=cols, yticklabels=rows)

print("The best homogeneity score of cosine UMAP is:", homogeneity_score(data_all.target, km_cos.labels_))
print("The best completeness_score score of cosine UMAP is:", completeness_score(data_all.target, km_cos.labels
print("The best V-measure score of cosine UMAP is:", v_measure_score(data_all.target, km_cos.labels_))
print("The best adjusted Rand Index of cosine UMAP score is:", adjusted_rand_score(data_all.target, km_cos.labe
print("The best adjusted mutual information of cosine UMAP score is:", adjusted_mutual_info_score(data_all.targ
```



```
The best homogeneity score of cosine UMAP is: 0.5745590392752127
The best completeness_score score of cosine UMAP is: 0.5955010889932821
The best V-measure score of cosine UMAP is: 0.5848426509253882
The best adjusted Rand Index of cosine UMAP score is: 0.46123389571290613
The best adjusted mutual information of cosine UMAP score is: 0.5834663368427103
```

Q11 Use UMAP to reduce the dimensionality of the 20 category TF-IDF matrix, and apply K-Means clustering with n_components=20 .

Results for both metric="euclidean" and metric="cosine" are shown below. After validated, Cosine performs much better than euclidean.

Question 12

Q12: Analyze the contingency matrix.

For euclidean, We can't see any pattern from the plot. There is no clear feature for the diagonal line, which indicates that this clusterer/ parameter performs poorly. The 5 clustering metrics values demonstrated this observation.

For cosine, we can definitely see a clear diagonal line, which indicates that there is a 'match' of the feature between corresponding rows and columns. While the values on diagonal are mainly around 600-800, the off-diagonal values are sometimes in unit digits. However, there are still mismatch elements lying beyond the diagonal, for example in column 12 and row 4. The off-diagonal elements contain the real information in most situations, and they are affecting the overall accuracy.

Question 13

Q13: 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.

UMAP (with metric="cosine") is the best so far. The second best is SVD(n_components=50) and I assume that NMF is better than sparse representation.

For UMAP, it has got homogeneity score of cosine UMAP: 0.5745590392752127 completeness_score score of cosine UMAP: 0.5955010889932821 V-measure score of cosine UMAP: 0.5848426509253882 adjusted Rand Index of cosine UMAP score: 0.46123389571290613 adjusted mutual information of cosine UMAP score: 0.5834663368427103

UMAP performs better in cosine. This is expected since cosine similarity is not affected by the magnitude of the vectors, meaning that the length of the documents does not affect the distance metric, whereas Euclidean distance might fail. UMAP is performing better than PCA and NMF. It is also reasonable because UMAP is a nonlinear dimensionality reduction method so that it is very effective for visualizing clusters.

Question 14

In [25]:
```python
from sklearn.cluster import AgglomerativeClustering

ward = AgglomerativeClustering(n_clusters=20, linkage='ward')
ward_cos = ward.fit(data_Q11_cos_r)
print("The Homogeneity score of Agglomerative Clustering Ward is:", homogeneity_score(data_all.target, ward_cos
print("The Completeness score of Agglomerative Clustering Ward is:", completeness_score(data_all.target, ward_c
print("The V-measure score of Agglomerative Clustering Ward is:", v_measure_score(data_all.target, ward_cos.lab
print("The adjusted rand-index score of Agglomerative Clustering Ward is:", adjusted_rand_score(data_all.target
print("The adjusted mutual information score of Agglomerative Clustering Ward is:", adjusted_mutual_info_score
print('\n')

single = AgglomerativeClustering(n_clusters=20, linkage='single')
single_cos = single.fit(data_Q11_cos_r)
print("The Homogeneity score of Agglomerative Clustering Single is:", homogeneity_score(data_all.target, single
print("The Completeness score of Agglomerative Clustering Single is:", completeness_score(data_all.target, sing
print("The V-measure score of Agglomerative Clustering Single is:", v_measure_score(data_all.target, single_cos
print("The adjusted rand-index score of Agglomerative Clustering Single is:", adjusted_rand_score(data_all.targ
print("The adjusted mutual information score of Agglomerative Clustering Single is:", adjusted_mutual_info_scor
```

```
The Homogeneity score of Agglomerative Clustering Ward is: 0.5558505524517964
The Completeness score of Agglomerative Clustering Ward is: 0.5787912934094087
The V-measure score of Agglomerative Clustering Ward is: 0.5670890093987669
The adjusted rand-index score of Agglomerative Clustering Ward is: 0.42957297903960145
The adjusted mutual information score of Agglomerative Clustering Ward is: 0.565650345747001


The Homogeneity score of Agglomerative Clustering Single is: 0.016239031063543
The Completeness score of Agglomerative Clustering Single is: 0.37270757888720096
The V-measure score of Agglomerative Clustering Single is: 0.031122060438750886
The adjusted rand-index score of Agglomerative Clustering Single is: 0.0004904152230090023
The adjusted mutual information score of Agglomerative Clustering Single is: 0.026331606809285767
```

Q14: Use UMAP to reduce the dimensionality properly, and perform Agglomerative clustering

Results are shown above.

Question 15

In [26]:
```python
#DBSCAN
from sklearn.cluster import DBSCAN
import hdbscan
UMAP_cos = data_Q11_cos_r
DBSCAN_ = DBSCAN(eps=0.5)
result_DBSCAN = DBSCAN_.fit_predict(UMAP_cos)
print("The Homogeneity score of DBSCAN is:", homogeneity_score(data_all.target, result_DBSCAN))
print("The Completeness score of DBSCAN  is:", completeness_score(data_all.target, result_DBSCAN))
print("The V-measure score of DBSCAN  is:", v_measure_score(data_all.target, result_DBSCAN))
print("The adjusted rand-index score of DBSCAN  is:", adjusted_rand_score(data_all.target, result_DBSCAN))
print("The adjusted mutual information score of DBSCAN is:", adjusted_mutual_info_score(data_all.target, result
print('\n')

#HDBSCAN
HDBSCAN_ = hdbscan.HDBSCAN(min_cluster_size=100)
result_HDBSCAN = HDBSCAN_.fit_predict(UMAP_cos)
print("The Homogeneity score of HDBSCAN is:", homogeneity_score(data_all.target, result_HDBSCAN))
print("The Completeness score of HDBSCAN is:", completeness_score(data_all.target, result_HDBSCAN))
print("The V-measure score of HDBSCAN is:", v_measure_score(data_all.target, result_HDBSCAN))
print("The adjusted rand-index score of HDBSCAN is:", adjusted_rand_score(data_all.target, result_HDBSCAN))
print("The adjusted mutual information score of HDBSCAN is:", adjusted_mutual_info_score(data_all.target, resul
```

```
The Homogeneity score of DBSCAN is: 0.017867202020543514
The Completeness score of DBSCAN  is: 0.3401863022760621
The V-measure score of DBSCAN  is: 0.033951224129637456
The adjusted rand-index score of DBSCAN  is: 0.0004932472884334998
The adjusted mutual information score of DBSCAN is: 0.028420528796731015


The Homogeneity score of HDBSCAN is: 0.4105371229231406
The Completeness score of HDBSCAN is: 0.6176158762091551
The V-measure score of HDBSCAN is: 0.493222789029545
The adjusted rand-index score of HDBSCAN is: 0.22281678126108806
The adjusted mutual information score of HDBSCAN is: 0.49229296936230066
```

Q15: Apply DBSCAN and HDBSCAN on UMAP-transformed 20-category data. Use min_cluster_size=100

Results are shown above

Question 16

In [27]:
```python
#HDBSCAN performs better

HDBSCAN_ = hdbscan.HDBSCAN(min_cluster_size=100)
result_HDBSCAN = HDBSCAN_.fit_predict(UMAP_cos)
cm = confusion_matrix(data_all.target, result_HDBSCAN)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], size=(15,15), xticklabels=cols, yticklabels=rows, title = 'HDBSCAN')
```

Q16 How many clusters are given by the model? What does "-1" mean for the clustering labels?

There are 10 clusters with values given by the model. There are a few ground truth clusters with value of 0. "-1" is the outlier or noise sample that have not been classified into the table. HDBSCAN are trying to find outlier, in which K-means can not identify.

Question 17

In [28]:
```python
from sklearn.pipeline import Pipeline
from tempfile import mkdtemp
from joblib import Memory
from shutil import rmtree

svd_5 = TruncatedSVD(n_components=5, random_state=0)
svd_20 = TruncatedSVD(n_components=20, random_state=0)
svd_200 = TruncatedSVD(n_components=200, random_state=0)
nmf_5 = NMF(n_components=5, max_iter = 500, random_state=0)
nmf_20 = NMF(n_components=20, max_iter = 500, random_state=0)
nmf_200 = NMF(n_components=200, max_iter = 500, random_state=0)
umap_cos_5 = umap.UMAP(n_components=5, metric='cosine', random_state=0)
umap_cos_20 = umap.UMAP(n_components=20, metric='cosine', random_state=0)
umap_cos_200 = umap.UMAP(n_components=200, metric='cosine', random_state=0)

km_10 = KMeans(n_clusters=10, random_state=0)
km_20 = KMeans(n_clusters=20, random_state=0)
km_50 = KMeans(n_clusters=50, random_state=0)
agg_ward_20 = AgglomerativeClustering(n_clusters=20)
DBSCAN_05 = DBSCAN(eps=0.5, n_jobs=-1)
DBSCAN_5 = DBSCAN(eps=5, n_jobs=-1)
HDBSCAN_100 = hdbscan.HDBSCAN(min_cluster_size=100, core_dist_n_jobs=-1)
HDBSCAN_200 = hdbscan.HDBSCAN(min_cluster_size=200, core_dist_n_jobs=-1)

dic = {
    'km10' : km_10,
    'km20' : km_20,
    'km50' : km_50,
    'agg_ward20' : agg_ward_20,
    'DBSCAN0.5' : DBSCAN_05,
    'DBSCAN5' : DBSCAN_5,
    'HDBSCAN100' : HDBSCAN_100,
    'HDBSCAN200' : HDBSCAN_200
    }
clustererlist = ['km10', 'km20', 'km50', 'agg_ward20','DBSCAN0.5', 'DBSCAN5', 'HDBSCAN100', 'HDBSCAN200']
lst = ['None', 'svd5', 'svd20', 'svd200', 'nmf5', 'nmf20', 'nmf200', 'umap5', 'umap20', 'umap200']

def pip_Q17(name):

    cachedir = mkdtemp()
    memory = Memory(location=cachedir, verbose=False)

    if name not in dic:
        return print('wrong name')
    clusterer_name = dic[name]

    preprocessors = [('none', None), ('svd5', svd_5),('svd20', svd_20), ('svd200', svd_200), ('nmf5', nmf_5),
     ('umap5', umap_cos_5), ('umap20', umap_cos_20), ('umap200', umap_cos_200)]

    clusterer = Pipeline([
    (name, clusterer_name,),]
    )

    #, memory=memory

    # None is not applicable for Agglomerative Clustering
    if name == 'agg_ward20':
        for i in range(len(preprocessors)-1):
            preprocessor_ = preprocessors[i+1]
            preprocessor  = Pipeline([
            preprocessor_]
            )
            pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer),])
            pipe.fit(data_all_tfidf)
            labels = pipe["clusterer"][name].labels_
            print("The adjusted Rand Index score for {} - {} is: {}".format(lst[i+1], name, adjusted_rand_score
    else:
        for i in range(len(preprocessors)):
            preprocessor_ = preprocessors[i]
            preprocessor  = Pipeline([
            preprocessor_]
            )
            pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer),])
            pipe.fit(data_all_tfidf)
            labels = pipe["clusterer"][name].labels_
            print("The adjusted Rand Index score for {} - {} is: {}".format(lst[i], name, adjusted_rand_score(

    rmtree(cachedir)
```

In [29]:
```python
import time

time_start =time.time()
for clusterer in clustererlist:
    pip_Q17(clusterer)
    print('\n')

print('Time to run code: {}'.format(time.time()-time_start))
```

```
The adjusted Rand Index score for None - km10 is: 0.10646082402966119
The adjusted Rand Index score for svd5 - km10 is: 0.104221694092446
The adjusted Rand Index score for svd20 - km10 is: 0.09345420336584254
The adjusted Rand Index score for svd200 - km10 is: 0.0918968655587027
The adjusted Rand Index score for nmf5 - km10 is: 0.0734015603795355
The adjusted Rand Index score for nmf20 - km10 is: 0.04298470526752116
The adjusted Rand Index score for nmf200 - km10 is: 0.008292090489404729
The adjusted Rand Index score for umap5 - km10 is: 0.33589654423963794
The adjusted Rand Index score for umap20 - km10 is: 0.3316531937559566
The adjusted Rand Index score for umap200 - km10 is: 0.3310964506148273


The adjusted Rand Index score for None - km20 is: 0.12216280815908323
The adjusted Rand Index score for svd5 - km20 is: 0.1264208062507061
The adjusted Rand Index score for svd20 - km20 is: 0.11418718267997306
The adjusted Rand Index score for svd200 - km20 is: 0.1062183531841657
The adjusted Rand Index score for nmf5 - km20 is: 0.08837736768788304
The adjusted Rand Index score for nmf20 - km20 is: 0.08374281597442834
The adjusted Rand Index score for nmf200 - km20 is: 0.00929401320087941
The adjusted Rand Index score for umap5 - km20 is: 0.4426323455482683
The adjusted Rand Index score for umap20 - km20 is: 0.43695516024122794
The adjusted Rand Index score for umap200 - km20 is: 0.43304613263257424


The adjusted Rand Index score for None - km50 is: 0.13610924306490127
The adjusted Rand Index score for svd5 - km50 is: 0.0982260466458153
The adjusted Rand Index score for svd20 - km50 is: 0.13601724422731648
The adjusted Rand Index score for svd200 - km50 is: 0.1178334341078419
The adjusted Rand Index score for nmf5 - km50 is: 0.07900787843196773
The adjusted Rand Index score for nmf20 - km50 is: 0.12089396000535328
The adjusted Rand Index score for nmf200 - km50 is: 0.016716267242620642
The adjusted Rand Index score for umap5 - km50 is: 0.3845691386597167
The adjusted Rand Index score for umap20 - km50 is: 0.36820480773425246
The adjusted Rand Index score for umap200 - km50 is: 0.39303475397252213


The adjusted Rand Index score for svd5 - agg_ward20 is: 0.11437723398830446
The adjusted Rand Index score for svd20 - agg_ward20 is: 0.20030091106433323
The adjusted Rand Index score for svd200 - agg_ward20 is: 0.11368851618049414
The adjusted Rand Index score for nmf5 - agg_ward20 is: 0.0858637876638969
The adjusted Rand Index score for nmf20 - agg_ward20 is: 0.10094123979015178
The adjusted Rand Index score for nmf200 - agg_ward20 is: 0.009165586826998327
The adjusted Rand Index score for umap5 - agg_ward20 is: 0.42957297903960145
The adjusted Rand Index score for umap20 - agg_ward20 is: 0.4193257188975031
The adjusted Rand Index score for umap200 - agg_ward20 is: 0.42996289436843965


The adjusted Rand Index score for None - DBSCAN0.5 is: -1.1025857555874216e-05
The adjusted Rand Index score for svd5 - DBSCAN0.5 is: 0.0
The adjusted Rand Index score for svd20 - DBSCAN0.5 is: 0.0
The adjusted Rand Index score for svd200 - DBSCAN0.5 is: -1.3945201645077132e-05
The adjusted Rand Index score for nmf5 - DBSCAN0.5 is: 0.0
The adjusted Rand Index score for nmf20 - DBSCAN0.5 is: 0.0
The adjusted Rand Index score for nmf200 - DBSCAN0.5 is: 0.0
The adjusted Rand Index score for umap5 - DBSCAN0.5 is: 0.0004932472884334998
The adjusted Rand Index score for umap20 - DBSCAN0.5 is: 0.10607153642918443
The adjusted Rand Index score for umap200 - DBSCAN0.5 is: 0.020097395582654044


The adjusted Rand Index score for None - DBSCAN5 is: 0.0
The adjusted Rand Index score for svd5 - DBSCAN5 is: 0.0
The adjusted Rand Index score for svd20 - DBSCAN5 is: 0.0
The adjusted Rand Index score for svd200 - DBSCAN5 is: 0.0
The adjusted Rand Index score for nmf5 - DBSCAN5 is: 0.0
The adjusted Rand Index score for nmf20 - DBSCAN5 is: 0.0
The adjusted Rand Index score for nmf200 - DBSCAN5 is: 0.0
The adjusted Rand Index score for umap5 - DBSCAN5 is: 2.7908288325089694e-06
The adjusted Rand Index score for umap20 - DBSCAN5 is: -3.4549557986687646e-07
The adjusted Rand Index score for umap200 - DBSCAN5 is: -3.4549557986687646e-07


The adjusted Rand Index score for None - HDBSCAN100 is: 0.0
The adjusted Rand Index score for svd5 - HDBSCAN100 is: 0.0
The adjusted Rand Index score for svd20 - HDBSCAN100 is: 0.0
The adjusted Rand Index score for svd200 - HDBSCAN100 is: 0.0
The adjusted Rand Index score for nmf5 - HDBSCAN100 is: 0.022363663425245085
The adjusted Rand Index score for nmf20 - HDBSCAN100 is: 0.0
The adjusted Rand Index score for nmf200 - HDBSCAN100 is: 0.0
The adjusted Rand Index score for umap5 - HDBSCAN100 is: 0.22281678126108806
The adjusted Rand Index score for umap20 - HDBSCAN100 is: 0.2004363239448477
The adjusted Rand Index score for umap200 - HDBSCAN100 is: 0.20841526062808333


The adjusted Rand Index score for None - HDBSCAN200 is: 0.0
The adjusted Rand Index score for svd5 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for svd20 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for svd200 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for nmf5 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for nmf20 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for nmf200 - HDBSCAN200 is: 0.0
The adjusted Rand Index score for umap5 - HDBSCAN200 is: 0.1928108415564108
The adjusted Rand Index score for umap20 - HDBSCAN200 is: 0.19912309075225604
The adjusted Rand Index score for umap200 - HDBSCAN200 is: 0.18494364521103032

 Time to run code: 14133.366367578506
```

Q17: Based on your experiments, which dimensionality reduction technique and clustering methods worked best together for 20-class text data and why?

reducer: UMAP5 clusterer: KMeans20 works the best, with a highest adjusted Rand Index score of 0.4426323455482683.

As the project instruction for Q17 states, we are only given limited parameter choices for each model, so I decide not to experiment on other parameters for each model besides the required parameters. UMAP is an exception, in which I use metric='cosine' since I have already figured out that it performs way better than the euclidean metric.

UMAP uses graph layout algorithms to arrange data in low-dimensional space. In the simplest sense, UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible. it is a nonlinear dimensionality reduction method, so that it might work better than PCA and it is very effective for visualizing clusters or groups of data points and their relative proximities. K-means can easily locate centroid and it can scale up for large amount of data.

Question 18

In [30]:
```python
#pip install ivis[cpu]
```

In [31]:
```python
# from ivis import Ivis
# from sklearn.preprocessing import MinMaxScaler

# model1 = Ivis(embedding_dims=2, k=15)
# model2 = Ivis(embedding_dims=10, k=15)
# model3 = Ivis(embedding_dims=20, k=15)
# model4 = Ivis(embedding_dims=2, k=50)
# model5 = Ivis(embedding_dims=10, k=50)
# model6 = Ivis(embedding_dims=20, k=50)

# km_10 = KMeans(n_clusters=10, random_state=0)
# km_20 = KMeans(n_clusters=20, random_state=0)
# km_50 = KMeans(n_clusters=50, random_state=0)
# agg_ward_20 = AgglomerativeClustering(n_clusters=20)
# DBSCAN_05 = DBSCAN(eps=0.5, n_jobs=-1)
# DBSCAN_5 = DBSCAN(eps=5, n_jobs=-1)
# HDBSCAN_100 = hdbscan.HDBSCAN(min_cluster_size=100, core_dist_n_jobs=-1)
# HDBSCAN_200 = hdbscan.HDBSCAN(min_cluster_size=200, core_dist_n_jobs=-1)

# dic = {
#     'km10' : km_10,
#     'km20' : km_20,
#     'km50' : km_50,
#     'agg_ward20' : agg_ward_20,
#     'DBSCAN0.5' : DBSCAN_05,
#     'DBSCAN5' : DBSCAN_5,
#     'HDBSCAN100' : HDBSCAN_100,
#     'HDBSCAN200' : HDBSCAN_200
#     }
# clustererlist = ['km10', 'km20', 'km50', 'agg_ward20','DBSCAN0.5', 'DBSCAN5', 'HDBSCAN100', 'HDBSCAN200']
# lst_Q18 = ['model1', 'model2', 'model3', 'model4', 'model5', 'model6']

# def pip_Q18(name):

#     cachedir = mkdtemp()
#     memory = Memory(location=cachedir, verbose=False)

#     if name not in dic:
#         return print('wrong name')
#     clusterer_name = dic[name]

#     preprocessors = [('model1',model1), ('model2', model2), ('model3', model3), ('model4', model4), ('model5

#     clusterer = Pipeline([
#     (name, clusterer_name,),]
#     )

#     #, memory=memory

#     for i in range(len(preprocessors)):
#         preprocessor_ = preprocessors[i]
#         preprocessor  = Pipeline([
#         preprocessor_]
#         )
#         pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer),])
#         data_all_tfidf_scaled = MinMaxScaler().fit_transform(data_all_tfidf)
#         pipe.fit(data_all_tfidf_scaled)
#         labels = pipe["clusterer"][name].labels_
#         print("The adjusted Rand Index score for {} - {} is: {}".format(lst_Q18[i], name, adjusted_rand_score

#     rmtree(cachedir)
```

In [32]:
```python
# time_start =time.time()
# for clusterer in clustererlist:
#     pip_Q18(clusterer)
#     print('\n')

# print('Time to run code: {}'.format(time.time()-time_start))
```

In [ ]:

Preparing data

In [306…
```python
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm
import requests
import os
import tarfile

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, adjusted_rand_score, adjusted_mutual_info_score
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin
```

Flowers Dataset and VGG Features

In [307…
```python
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:
    if not os.path.exists('./flower_photos'):
        # download the flowers dataset and extract its images
        url = 'http://download.tensorflow.org/example_images/flower_photos.tgz'
        with open('./flower_photos.tgz', 'wb') as file:
            file.write(requests.get(url).content)
        with tarfile.open('./flower_photos.tgz') as file:
            file.extractall('./')
        os.remove('./flower_photos.tgz')

    class FeatureExtractor(nn.Module):
        def __init__(self):
            super().__init__()

            vgg = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=True)

            # Extract VGG-16 Feature Layers
            self.features = list(vgg.features)
            self.features = nn.Sequential(*self.features)
            # Extract VGG-16 Average Pooling Layer
            self.pooling = vgg.avgpool
            # Convert the image into one-dimensional vector
            self.flatten = nn.Flatten()
            # Extract the first part of fully-connected layer from VGG16
            self.fc = vgg.classifier[0]

        def forward(self, x):
            # It will take the input 'x' until it returns the feature vector called 'out'
            out = self.features(x)
            out = self.pooling(out)
            out = self.flatten(out)
            out = self.fc(out)
            return out

    # Initialize the model
    assert torch.cuda.is_available()
    feature_extractor = FeatureExtractor().cuda().eval()

    dataset = datasets.ImageFolder(root='./flower_photos',
                                   transform=transforms.Compose([transforms.Resize(224),
                                                                 transforms.CenterCrop(224),
                                                                 transforms.ToTensor(),
                                                                 transforms.Normalize(mean=[0.485, 0.456, 0.406
    dataloader = DataLoader(dataset, batch_size=64, shuffle=True)

    # Extract features and store them on disk
    f_all, y_all = np.zeros((0, 4096)), np.zeros((0,))
    for x, y in tqdm(dataloader):
        with torch.no_grad():
            f_all = np.vstack([f_all, feature_extractor(x.cuda()).cpu()])
            y_all = np.concatenate([y_all, y])
    np.savez(filename, f_all=f_all, y_all=y_all)
```

In [308…
```python
print(f_all.shape, y_all.shape)
num_features = f_all.shape[1]
num_features
```

```
(3670, 4096) (3670,)
```

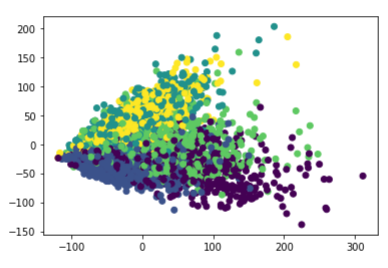
Out[308… 4096

In [309…
```python
type(f_all)
```

Out[309… numpy.ndarray

In [310…
```python
f_pca = PCA(n_components=2).fit_transform(f_all)
plt.scatter(*f_pca.T, c=y_all)
```

Out[310… <matplotlib.collections.PathCollection at 0x7f2cf067bb90>



MLP Classifier

In [311…
```python
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()
        #self.model.eval()
        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)
        running_corrects = 0
        epoch_acc = 0
        for epoch in tqdm(range(100)):
            for (X_, y_) in dataloader:
                ##################################
                # you should implement this part #
                ##################################
                optimizer.zero_grad()
                output = self(X_)
                loss = criterion(output, y_)
                loss.backward()
                optimizer.step()

        return self

    def eval(self, X_test, y_test):
        ##################################
        # you should implement this part #
        ##################################
        criterion = nn.NLLLoss()
        X_test = torch.tensor(X_test, dtype=torch.float32, device='cuda')
        y_test = torch.tensor(y_test, dtype=torch.int64, device='cuda')
        self.model.eval()
        dataset = TensorDataset(X_test, y_test)
        dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

        correct = 0
        total = 0
        acc = 0
        with torch.no_grad():
            for (X_, y_) in dataloader:
                output = self(X_)
                #loss = criterion(output, y_)
                #print('Epoch {}: train loss: {}'.format(epoch, loss.item()))
                pred = output.max(1, keepdim=True)[1]
                correct += pred.eq(y_.view_as(pred)).sum().item()
                total += X_.shape[0]
                acc = correct / total

        print('Average accuracy is: {}'.format(acc))

        #raise NotImplementedError
```

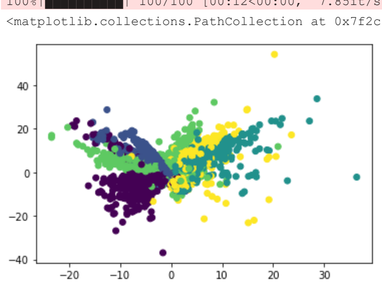
Autoencoder

In [312…
```python
class Autoencoder(torch.nn.Module, TransformerMixin):
    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()
```

In [313…
```python
X_em =Autoencoder(2).fit_transform(f_all)
plt.scatter(*X_em.T, c=y_all)
```

```
100%|██████████| 100/100 [00:12<00:00,  7.85it/s]
```

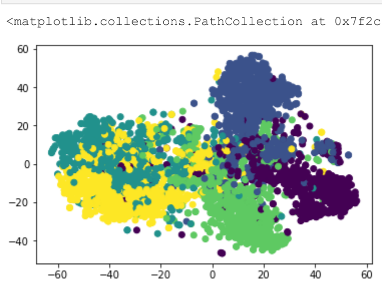
Out[313… <matplotlib.collections.PathCollection at 0x7f2cf0976410>



TSNE

In [314…
```python
from sklearn.manifold import TSNE
f_tsne = TSNE(n_components=2, learning_rate='auto', init='random').fit_transform(f_all)
plt.scatter(*f_tsne.T, c=y_all)
```

Out[314… <matplotlib.collections.PathCollection at 0x7f2cf02a8650>



Question 24

In [315…
```python
#pip install umap-learn
```

In [316…
```python
#pip install hdbscan --no-build-isolation --no-binary :all:
```

In [317…
```python
from sklearn.decomposition import TruncatedSVD
import umap.umap_ as umap
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
import hdbscan
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.pipeline import Pipeline

svd = TruncatedSVD(n_components=50)
umap_ = umap.UMAP(metric='cosine', n_components=50)
umap_ = umap.UMAP(n_components=50)
auto =Autoencoder(50)

km = KMeans(n_clusters=5)
agg = AgglomerativeClustering(n_clusters=5)
hdbs = hdbscan.HDBSCAN(min_cluster_size=170, min_samples=5)


preprocessors = Pipeline(
[('none', None),('svd', svd),('umap', umap_),('auto', auto),]
)
clusterer1 = Pipeline([
("km", km,),]
)
clusterer2 = Pipeline([
("agg", agg,),]
)
clusterer3 = Pipeline([
("hdbs", hdbs,),]
)

lst = ['None', 'svd', 'umap', 'auto']
for i in range(len(preprocessors)):
  preprocessor = preprocessors[i]
  pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer1),])
  pipe.fit(f_all)
  labels = pipe["clusterer"]["km"].labels_
  print("The adjusted Rand Index score for {}-km is: {}".format(lst[i], adjusted_rand_score(y_all, labels)))

for i in range(len(preprocessors)):
  preprocessor = preprocessors[i]
  pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer2),])
  pipe.fit(f_all)
  labels = pipe["clusterer"]["agg"].labels_
  print("The adjusted Rand Index score for {}-agg is: {}".format(lst[i], adjusted_rand_score(y_all, labels)))

for i in range(len(preprocessors)):
  preprocessor = preprocessors[i]
  pipe = Pipeline([("preprocessor", preprocessor),("clusterer", clusterer3),])
  pipe.fit(f_all)
  labels = pipe["clusterer"]["hdbs"].labels_
  print("The adjusted Rand Index score for {}-HDBSCAN is: {}".format(lst[i], adjusted_rand_score(y_all, labels)
```

```
The adjusted Rand Index score for None-km is: 0.18919803381799868
The adjusted Rand Index score for svd-km is: 0.19331418758222305
The adjusted Rand Index score for umap-km is: 0.42033491987510535
100%|██████████| 100/100 [00:12<00:00,  7.87it/s]
The adjusted Rand Index score for auto-km is: 0.20043596300242858
The adjusted Rand Index score for None-agg is: 0.18855278251971858
The adjusted Rand Index score for svd-agg is: 0.2673694864565261
The adjusted Rand Index score for umap-agg is: 0.3843379511508335
100%|██████████| 100/100 [00:12<00:00,  7.87it/s]
The adjusted Rand Index score for auto-agg is: 0.2517300185890039
The adjusted Rand Index score for None-HDBSCAN is: 0.0
The adjusted Rand Index score for svd-HDBSCAN is: 0.0
The adjusted Rand Index score for umap-HDBSCAN is: 0.34823802703770457
100%|██████████| 100/100 [00:12<00:00,  7.87it/s]
The adjusted Rand Index score for auto-HDBSCAN is: 0.0
```

Question 25

In [318…
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(f_all, y_all, test_size = 0.2, random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2936, 4096)
(734, 4096)
(2936,)
(734,)
```

In [319…
```python
model1 = MLP(num_features)
model1.train(X_train, y_train)
model1.eval(X_test, y_test)
```

```
100%|██████████| 100/100 [00:04<00:00, 20.71it/s]
Average accuracy is: 0.9032697547683923
```

In [320…
```python
#using n_components=50
umap_ = umap.UMAP(metric='cosine', n_components=50)
svd = TruncatedSVD(n_components=50)
auto =Autoencoder(50)

X_train_umap = umap_.fit_transform(X_train)
X_test_umap = umap_.transform(X_test)
X_train_svd = svd.fit_transform(X_train)
X_test_svd = svd.transform(X_test)
X_train_auto = auto.fit_transform(X_train)
X_test_auto = auto.transform(X_test)

# print(X_train_umap.shape)
# print(X_test_umap.shape)

num = 50
model2 = MLP(num)

print('\nUMAP')
model2.train(X_train_umap, y_train)
model2.eval(X_test_umap, y_test)

print('\nSVD')
model2.train(X_train_svd, y_train)
model2.eval(X_test_svd, y_test)

print('\nAUTO')
model2.train(X_train_auto, y_train)
model2.eval(X_test_auto, y_test)
```

```
100%|██████████| 100/100 [00:10<00:00,  9.92it/s]
UMAP
100%|██████████| 100/100 [00:04<00:00, 20.71it/s]
Average accuracy is: 0.8596730245231607

SVD
100%|██████████| 100/100 [00:04<00:00, 20.79it/s]
Average accuracy is: 0.8841961852861036

AUTO
100%|██████████| 100/100 [00:04<00:00, 20.65it/s]
Average accuracy is: 0.8474114441416893
```

written answers

Q19. 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?

In practice, very few people train an entire VGG Network from scratch. Instead, it is common to pre-train on a very large dataset, for example ImageNet, which contains 1.2 million images with 1000 categories, and then use the VGG network either as an initialization or a fixed feature extractor for the task of interest. It is the method of transfer learning.

Q20. In a brief paragraph explain how the helper code base is performing feature extraction.

It first loads a pre-trained network from ImageNet dataset. This code is using VGG-16 (model of the 16-layer network (vgg16)) as the network. It then defines a new, untrained feed-forward network as a classifier. In VGG16 it mainly has three parts:

Convolution layer- In this layer, filters are applied to extract features from images.

Pooling layer- Its function is to reduce the spatial size.

Fully Connected- These are fully connected connections to the previous layers.

Q21 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?

The original pictures are '320x240', '500x333', '500x318', '231x240' and so on. Images are croped into size 224 x 224 pixels. From the code, I should get each image a vector with 4096 features. And the shape for orginal VGG is (3670, 4096)

Q22: Are the extracted features dense or sparse? (Compare with sparse TF-IDF features in text.)

Dense. VGG16 can expand into a vector with much less channels comparing with TF-IDF.

Q23: 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.

The plot shows a two-dimensional visualization of the data. The colors define the target objects and their feature data location in 2D space. Since there might be dataset distributed in non-linearly form, t-SNE performs much better in clustering comparing with PCA. The clustering performance can be found to be more obvious and distinguishing in compare with the PCA.

Q24: 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.

HDBSCAN: min cluster size = 170 and min samples = 5

The best one is UMAP and K-Means, which has got the adjusted Rand Index score of 42%

Q25: 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.

The performance of the model suffer with the reduced-dimension representations. I test on SVD, UMAP and Autoencoder. The accuracy before and after dimensionality methods decrease, however, not significant, as the accuracy before and after are all around 85-90%. The reason it that MLP does not rely on low dimensionanal features. The MLP depends on input layer weights of the neural network, which has a simlar functionality as dimensionality reduction. MLP, as a neural network, performs better than results from Question 24. Thats is because MLP can work better in a more complex case. MLP is using multiple layers and each layer is adding its own level of non-linearity in a feed-forward multi-layer perceptron.