Student details

NAME : Vignesh Nagarajan

UID: 606185377

In [1]:
```
%cd '/content/drive/MyDrive/219/Project2'
```

```
/content/drive/MyDrive/219/Project2
```

In [2]:
```
!pip uninstall umap
!pip install umap-learn
```

```
WARNING: Skipping umap as it is not installed.
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.9/90.9 kB 1.2 MB/
s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from u
map-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from
umap-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (1.2.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-packages (from
umap-learn) (0.58.1)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.8/55.8 kB 5.5 MB/
s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from umap-lea
rn) (4.66.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-p
ackages (from numba>=0.51.2->umap-learn) (0.41.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from
pynndescent>=0.5->umap-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn>=0.22->umap-learn) (3.2.0)
Building wheels for collected packages: umap-learn
  Building wheel for umap-learn (setup.py) ... done
  Created wheel for umap-learn: filename=umap_learn-0.5.5-py3-none-any.whl size=86832 sha256=
53c92e248c550a93cbaaa82512656ba2d8d542a24d26858afd4f4fe899952fad
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a3b431db59b806a10da73661
2ebbc66c1bcc5
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
```

In [3]:
```
!pip install hdbscan
```

```
Collecting hdbscan
  Downloading hdbscan-0.8.33.tar.gz (5.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.2/5.2 MB 10.5 MB/s
eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting cython<3,>=0.27 (from hdbscan)
  Using cached Cython-0.29.37-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinu
x_2_24_x86_64.whl (1.9 MB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from h
dbscan) (1.23.5)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from hd
bscan) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dist-packages
(from hdbscan) (1.2.2)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.10/dist-packages (from h
dbscan) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn>=0.20->hdbscan) (3.2.0)
Building wheels for collected packages: hdbscan
  Building wheel for hdbscan (pyproject.toml) ... done
  Created wheel for hdbscan: filename=hdbscan-0.8.33-cp310-cp310-linux_x86_64.whl size=303927
8 sha256=1ff7286131e48eab520fd0a4b87810a22fcb1c8e6cac4a3c1a3c33235125d5cb
  Stored in directory: /root/.cache/pip/wheels/75/0b/3b/dc4f60b7cc455efaefb62883a7483e76f09d0
6ca81cf87d610
Successfully built hdbscan
Installing collected packages: cython, hdbscan
  Attempting uninstall: cython
    Found existing installation: Cython 3.0.8
    Uninstalling Cython-3.0.8:
      Successfully uninstalled Cython-3.0.8
Successfully installed cython-0.29.37 hdbscan-0.8.33
```

In [4]:

```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import re
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfTransformer
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
from sklearn import svm
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score, roc_curv
e, auc
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV, cross_val_score, GridSearchCV
from scipy.stats import uniform
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier
import warnings
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.pipeline import Pipeline
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.datasets import fetch_20newsgroups
nltk.download('punkt')#, if you need "tokenizers/punkt/english.pickle", choose it
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.metrics.cluster import homogeneity_score,\
                                    completeness_score,\
                                    v_measure_score,\
                                    adjusted_rand_score,\
                                    adjusted_mutual_info_score
from sklearn.cluster import KMeans,  AgglomerativeClustering, DBSCAN
from tqdm import tqdm
import umap.umap_ as umap
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import contingency_matrix
import itertools
import matplotlib.colors as colors
```

```
from plotmat import plot_mat
import hdbscan
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

2-class

Dataset feature extraction

In [ ]:
```
categories = ['comp.sys.ibm.pc.hardware', 'comp.graphics',
'comp.sys.mac.hardware', 'comp.os.ms-windows.misc',
'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey']
df = fetch_20newsgroups(subset='all', categories=categories,
shuffle=True, random_state=42)

#binary label separated , comp = 0 rec = 1
def label_gen(data):
    labels = []
    new_name = ['Computer','Recreation']

    for label in data.target:
        name = data.target_names[label]
        if name.startswith('comp'):
            labels.append(0)
        elif name.startswith('rec'):
            labels.append(1)
    return labels, new_name

label, label_name = label_gen(df)

print(np.unique(label))
print(label_name)
```

```
[0 1]
['Computer', 'Recreation']
```

In [ ]:
```
vec = CountVectorizer(min_df=3,stop_words='english')
dvec = vec.fit_transform(df.data)

tfidf = TfidfTransformer()
data_tfidf = tfidf.fit_transform(dvec)
```

In [ ]:
```
print(data_tfidf.shape)
```

```
(7882, 27768)
```

ANSWER-1

The tfidf matrix shape is (7882, 27768)

Contingency matrix K-means

In [ ]:
```
km = KMeans(n_clusters=2, max_iter=1500, n_init=30, random_state=42)
km.fit(data_tfidf)
print(contingency_matrix(label, km.labels_))
```

```
[[   4 3899]
 [1717 2262]]
```

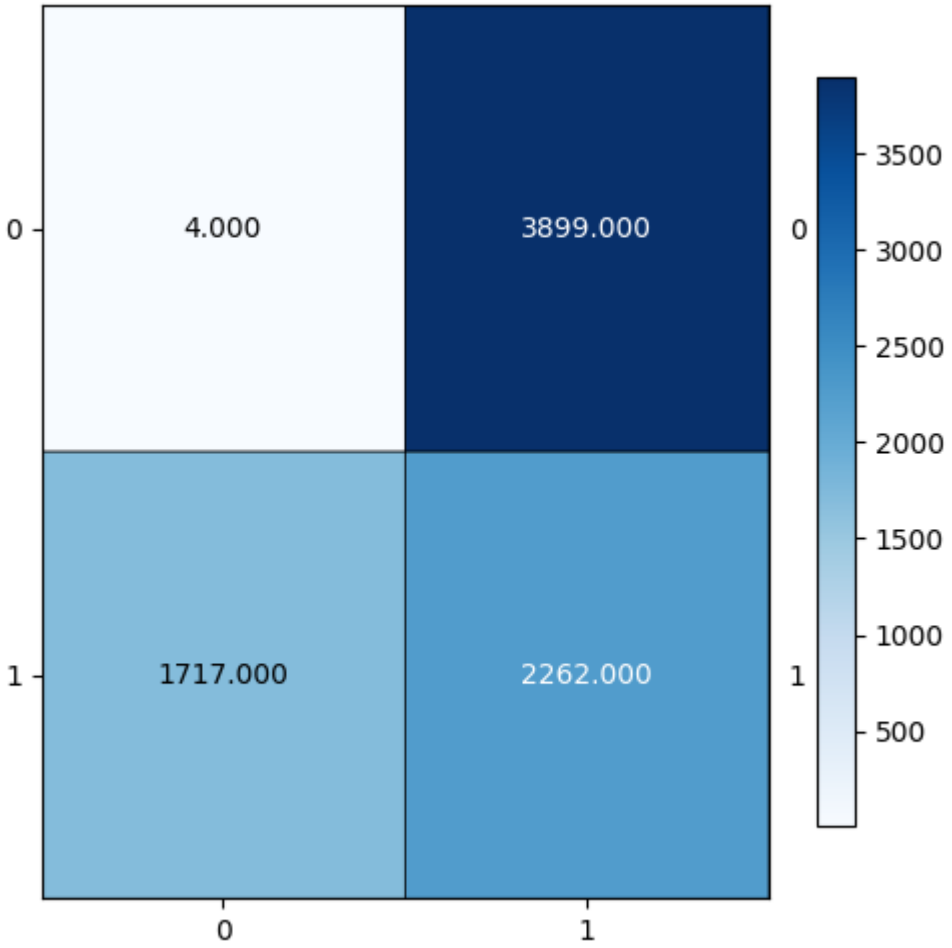
In [ ]:

```
def plot_mat(mat, xticklabels = None, yticklabels = None, pic_fname = None, size=(-1,-1), if
_show_values = True,
             colorbar = True, grid = 'k', xlabel = None, ylabel = None, title = None, vmin=N
one, vmax=None):
    if size == (-1, -1):
        size = (mat.shape[1] / 3, mat.shape[0] / 3)

    fig = plt.figure(figsize=size)
    ax = fig.add_subplot(1,1,1)

    # im = ax.imshow(mat, cmap=plt.cm.Blues)
    im = ax.pcolor(mat, cmap=plt.cm.Blues, linestyle='-', linewidth=0.5, edgecolor=grid, vmi
n=vmin, vmax=vmax)

    if colorbar:
        plt.colorbar(im,fraction=0.046, pad=0.06)
    # tick_marks = np.arange(len(classes))
    # Ticks
    lda_num_topics = mat.shape[0]
    nmf_num_topics = mat.shape[1]
    yticks = np.arange(lda_num_topics)
    xticks = np.arange(nmf_num_topics)
    ax.set_xticks(xticks + 0.5)
    ax.set_yticks(yticks + 0.5)
    if xticklabels is None:
        xticklabels = [str(i) for i in xticks]
    if yticklabels is None:
        yticklabels = [str(i) for i in yticks]
    ax.set_xticklabels(xticklabels)
    ax.set_yticklabels(yticklabels)

    # Minor ticks
    # ax.set_xticks(xticks, minor=True);
    # ax.set_yticks(yticks, minor=True);
    # ax.set_xticklabels([], minor=True)
    # ax.set_yticklabels([], minor=True)

    # ax.grid(which='minor', color='k', linestyle='-', linewidth=0.5)

    # tick labels on all four sides
    ax.tick_params(labelright = True, labeltop = False)

    if ylabel:
        plt.ylabel(ylabel, fontsize=15)
    if xlabel:
        plt.xlabel(xlabel, fontsize=15)
```

```
    if title:
        plt.title(title, fontsize=15)

    # im = ax.imshow(mat, interpolation='nearest', cmap=plt.cm.Blues)
    ax.invert_yaxis()

    # thresh = mat.max() / 2

    def show_values(pc, fmt="%.3f", **kw):
        pc.update_scalarmappable()
        ax = pc.axes
        for p, color, value in itertools.zip_longest(pc.get_paths(), pc.get_facecolors(), p
c.get_array()):
            x, y = p.vertices[:-2, :].mean(0)
            if np.all(color[:3] > 0.5):
                color = (0.0, 0.0, 0.0)
            else:
                color = (1.0, 1.0, 1.0)
            ax.text(x, y, fmt % value, ha="center", va="center", color=color, **kw, fontsize
=10)

    if if_show_values:
        show_values(im)
    # for i, j in itertools.product(range(mat.shape[0]), range(mat.shape[1])):
    #     ax.text(j, i, "{:.2f}".format(mat[i, j]), fontsize = 4,
    #             horizontalalignment="center",
    #             color="white" if mat[i, j] > thresh else "black")

    plt.tight_layout()
    if pic_fname:
        plt.savefig(pic_fname, dpi=300, transparent=True)
    plt.show()
    plt.close()
```

ANSWER-2

In [ ]:
```
plot_mat(contingency_matrix(label, km.labels_),size = (5,5))
```



In [ ]:
```
print("Homogeneity score :",
      homogeneity_score(label, km.labels_))
print("Completeness score:",
      completeness_score(label, km.labels_))
print("V-measure score :",
      v_measure_score(label, km.labels_))
print("Adjusted Rand Index score :",
      adjusted_rand_score(label, km.labels_))
print("Adjusted mutual information score :",
      adjusted_mutual_info_score(label, km.labels_))
```

```
Homogeneity score : 0.25341287993596523
Completeness score: 0.3346772445110614
V-measure score : 0.2884303641736151
Adjusted Rand Index score : 0.18054602343005735
Adjusted mutual information score : 0.2883562081995212
```

Answer-3

Metric	Score
Homogeneity Score	0.253
Completeness Score	0.335
V-measure Score	0.288
Adjusted Rand Index Score	0.181
Adjusted Mutual Information Score	0.288

Clustering with dense text representations

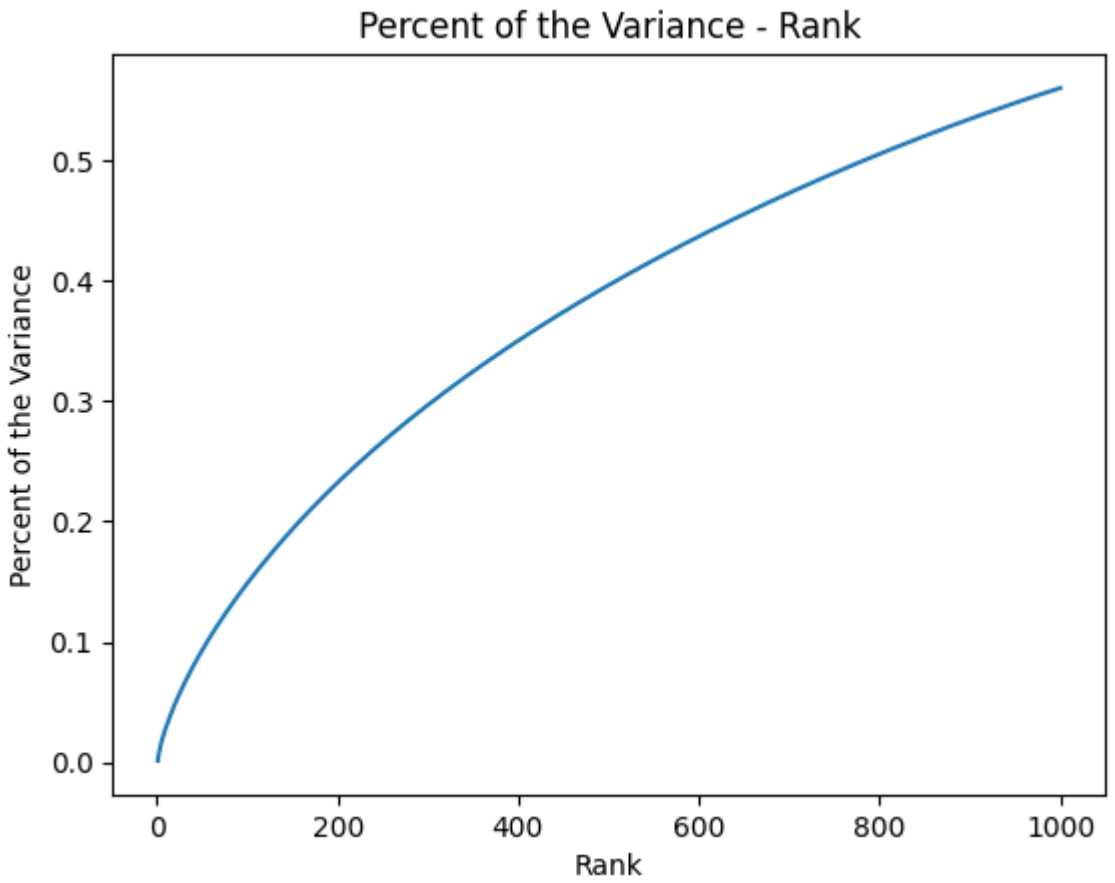
Answer-4

The plots are as shown in below cells for SVD and NMF. I've also included a close-up zoomed in view for choosing best rank 'r'.

SVD

In [ ]:
```
r = 1000
svd = TruncatedSVD(1000)
svd.fit(data_tfidf)
ratio = svd.explained_variance_ratio_.cumsum()
x = np.array(range(1, 1001))

plt.plot(x, ratio)
plt.title('Percent of the Variance - Rank')
plt.xlabel('Rank')
plt.ylabel('Percent of the Variance')
```

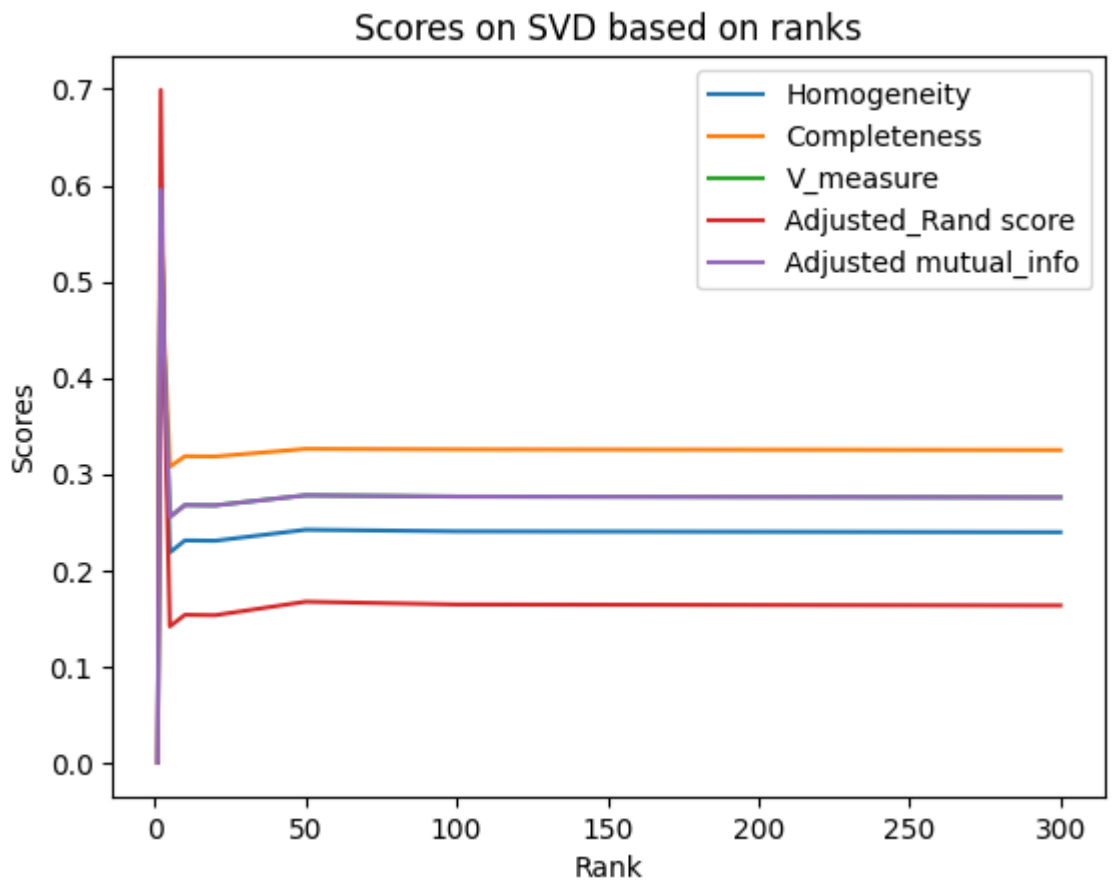
Out[ ]: Text(0, 0.5, 'Percent of the Variance')



In [ ]:
```
# sweep over different ranks using svd
ranks = [1, 2, 3, 5, 10, 20, 50, 100, 300]
h,c,v,ar,ami = [],[],[],[],[]
for rank in ranks:
    svd = TruncatedSVD(n_components=rank)
    data_svdr = svd.fit_transform(data_tfidf)
    km.fit(data_svdr)
    h.append(homogeneity_score(label,km.labels_))
    c.append(completeness_score(label,km.labels_))
    v.append(v_measure_score(label,km.labels_))
    ar.append(adjusted_rand_score(label,km.labels_))
    ami.append(adjusted_mutual_info_score(label,km.labels_))
```

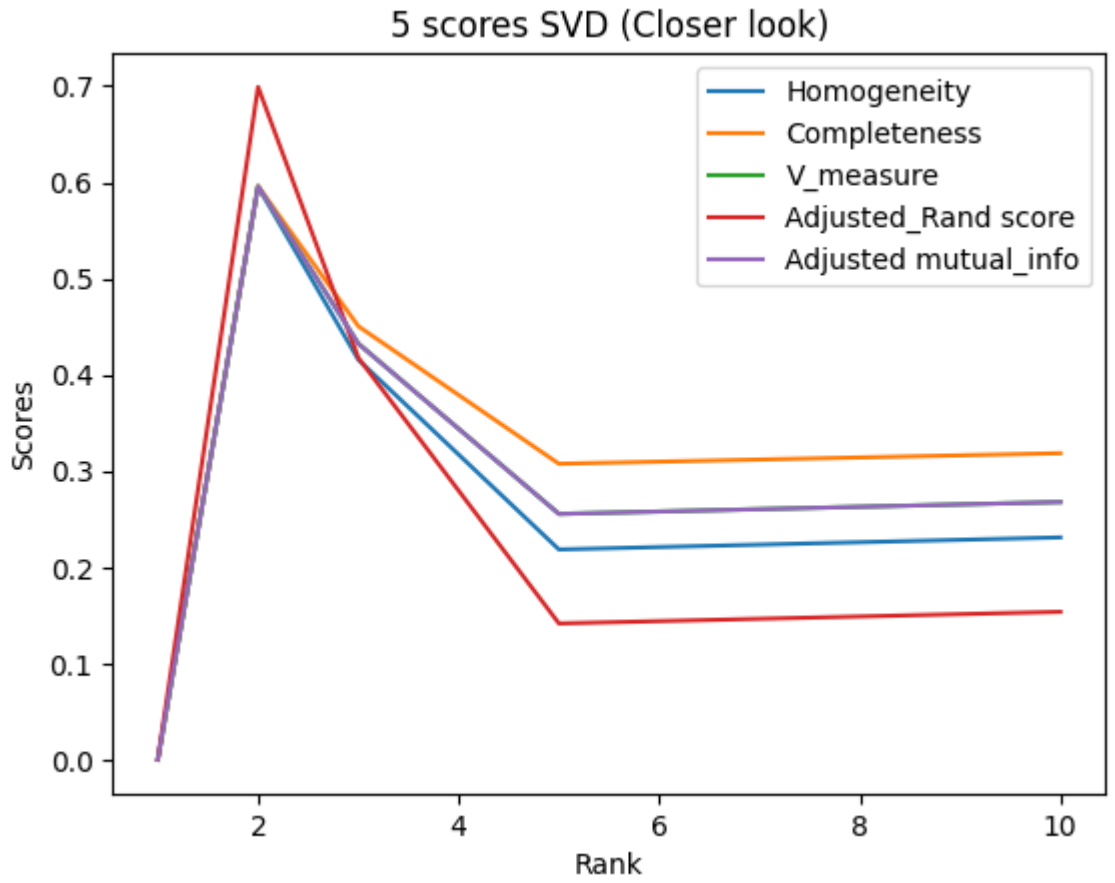
In [ ]:
```
plt.plot(ranks, h)
plt.plot(ranks, c)
plt.plot(ranks, v)
plt.plot(ranks, ar)
plt.plot(ranks, ami)
plt.title('Scores on SVD based on ranks')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f73f7fbb670>
```



In [ ]:
```
plt.plot(ranks[:5], h[:5])
plt.plot(ranks[:5], c[:5])
plt.plot(ranks[:5], v[:5])
plt.plot(ranks[:5], ar[:5])
plt.plot(ranks[:5], ami[:5])
plt.title('5 scores SVD (Closer look)')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f73f7faf310>
```



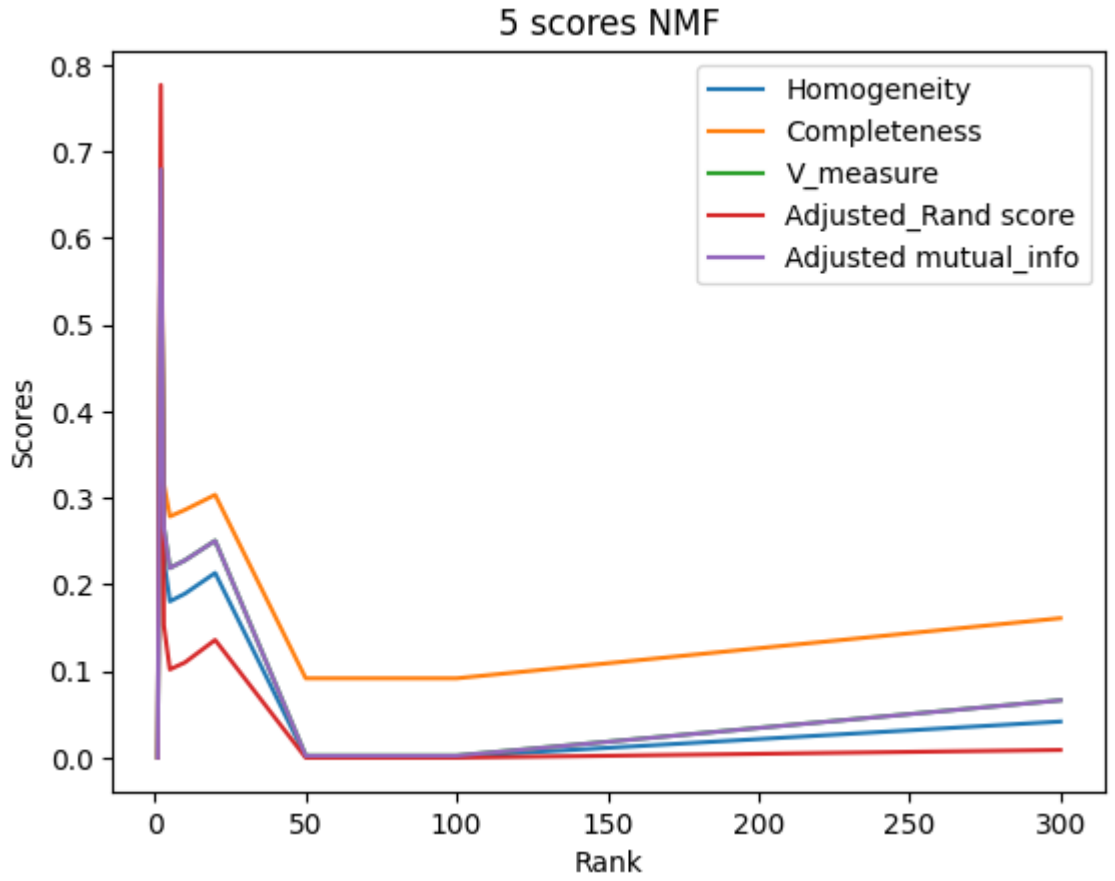
The best rank from all the scores is clearly r=2

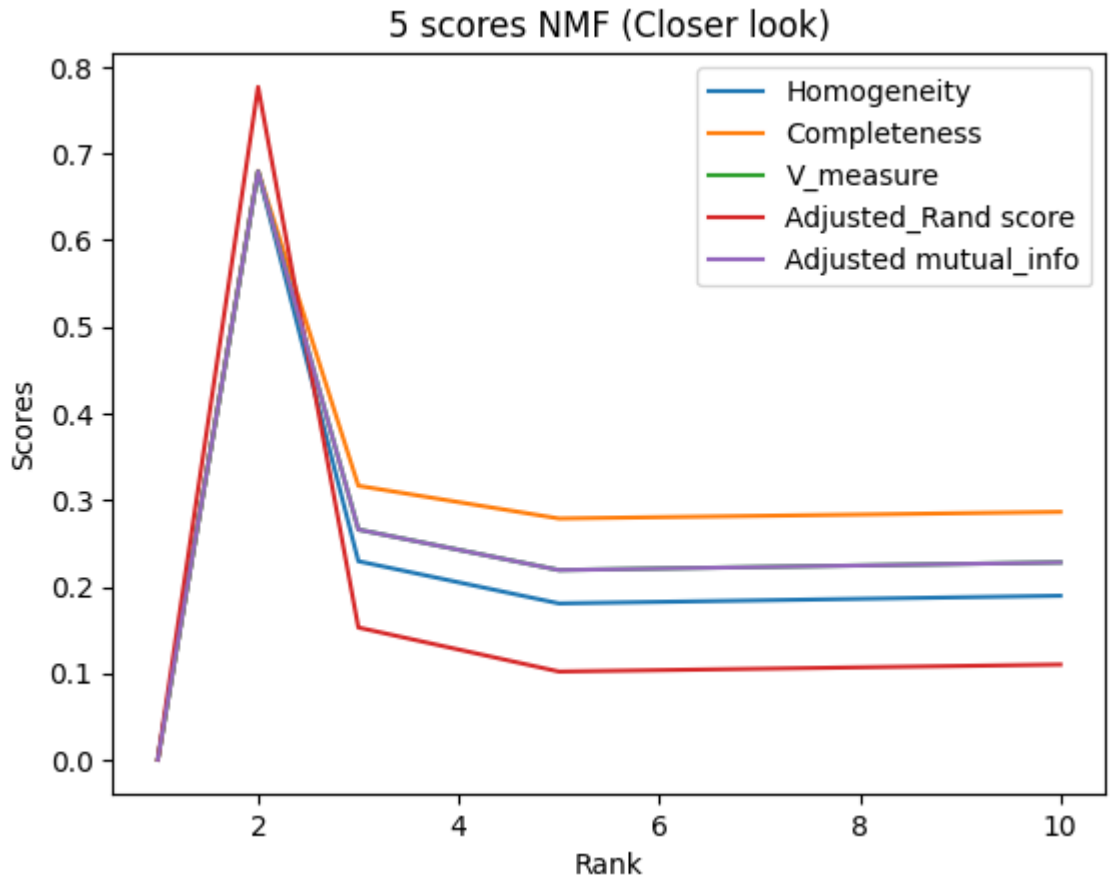
NMF

In [ ]:

```
# process data using nmf
from sklearn.decomposition import NMF

ranks = [1, 2, 3, 5, 10, 20, 50, 100, 300]
h_nmf,c_nmf,v_nmf,ar_nmf,ami_nmf = [],[],[],[],[]
for rank in ranks:
    nmf = NMF(n_components=rank)
    data_nmf_r = nmf.fit_transform(data_tfidf)
    km.fit(data_nmf_r)
    h_nmf.append(homogeneity_score(label,km.labels_))
    c_nmf.append(completeness_score(label,km.labels_))
    v_nmf.append(v_measure_score(label,km.labels_))
    ar_nmf.append(adjusted_rand_score(label,km.labels_))
    ami_nmf.append(adjusted_mutual_info_score(label,km.labels_))

plt.plot(ranks, h_nmf)
plt.plot(ranks, c_nmf)
plt.plot(ranks, v_nmf)
plt.plot(ranks, ar_nmf)
plt.plot(ranks, ami_nmf)
plt.title('5 scores NMF')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
# plot within 20
plt.plot(ranks[:5], h_nmf[:5])
plt.plot(ranks[:5], c_nmf[:5])
plt.plot(ranks[:5], v_nmf[:5])
plt.plot(ranks[:5], ar_nmf[:5])
plt.plot(ranks[:5], ami_nmf[:5])
plt.title('5 scores NMF (Closer look)')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWarnin
g: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```


5 scores NMF
Homogeneity
Completeness
V_measure
Adjusted_Rand score
Adjusted mutual_info
Scores
Rank
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
50
100
150
200
250
300



ANSWER 5

As seen from the plots above, the best case r = 2 for both PCA and NMF.

ANSWER 6

The reason why the non-monotonic behaviour is observed is owing to the "Curse of Dimensionality". As the number of dimensions increases, the distance between points in the high-dimensional space tends to become more uniform, and this can make it challenging for k-means to effectively cluster data in higher-dimensional spaces.

In [ ]:
```
print("Homogeneity score :",
      h_nmf[1])
print("Completeness score:",
      c_nmf[1])
print("V-measure score :",
      v_nmf[1])
print("Adjusted Rand Index score :",
      ar_nmf[1])
print("Adjusted mutual information score :",
      ami_nmf[1])
```

```
Homogeneity score : 0.6790483562300399
Completeness score: 0.680131609210451
V-measure score : 0.6795895510492934
Adjusted Rand Index score : 0.7770177788377391
Adjusted mutual information score : 0.6795601939855643
```

ANSWER 7

Clearly, the results on average on all 5 scores are much better than those obtained for Q3. The 5 scores for r=2 are as follows:

Metric	Score
Homogeneity Score	0.679
Completeness Score	0.680
V-measure Score	0.680
Adjusted Rand Index Score	0.777
Adjusted Mutual Information Score	0.680

We see that these results are around thrice as obtained in Q3

Visualize Clusters

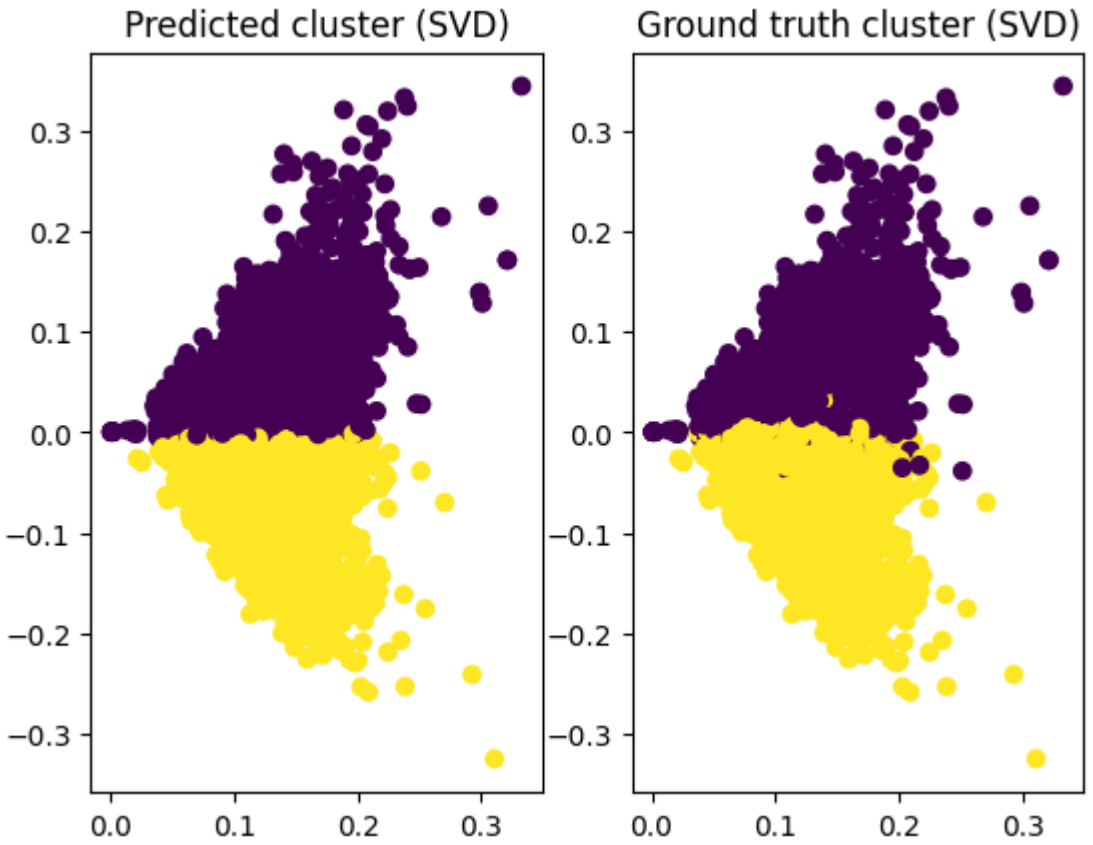
ANSWER-8

Plots are as shown below for best 'r' in SVM and NMF obtained from Answer-5

In [ ]:

```
svd = TruncatedSVD(n_components=2)
data_svd_2 = svd.fit_transform(data_tfidf)
nmf = NMF(n_components=2)
data_nmf_2 = nmf.fit_transform(data_tfidf)
#  svd
plt.figure(figsize=(6, 6))
km.fit(data_svd_2)

plt.subplot(121)
plt.scatter(data_svd_2[:, 0], data_svd_2[:, 1], c=km.labels_)
plt.title("Predicted cluster (SVD)")

plt.subplot(122)
plt.scatter(data_svd_2[:, 0], data_svd_2[:, 1], c=label)
plt.title("Ground truth cluster (SVD)")

print("Homogeneity score :",
      homogeneity_score(label, km.labels_))
print("Completeness_score score :",
      completeness_score(label, km.labels_))
print("V-measure score is:",
      v_measure_score(label, km.labels_))
print("Adjusted Rand Index score :",
      adjusted_rand_score(label, km.labels_))
print("Adjusted mutual information score :",
      adjusted_mutual_info_score(label, km.labels_))
```

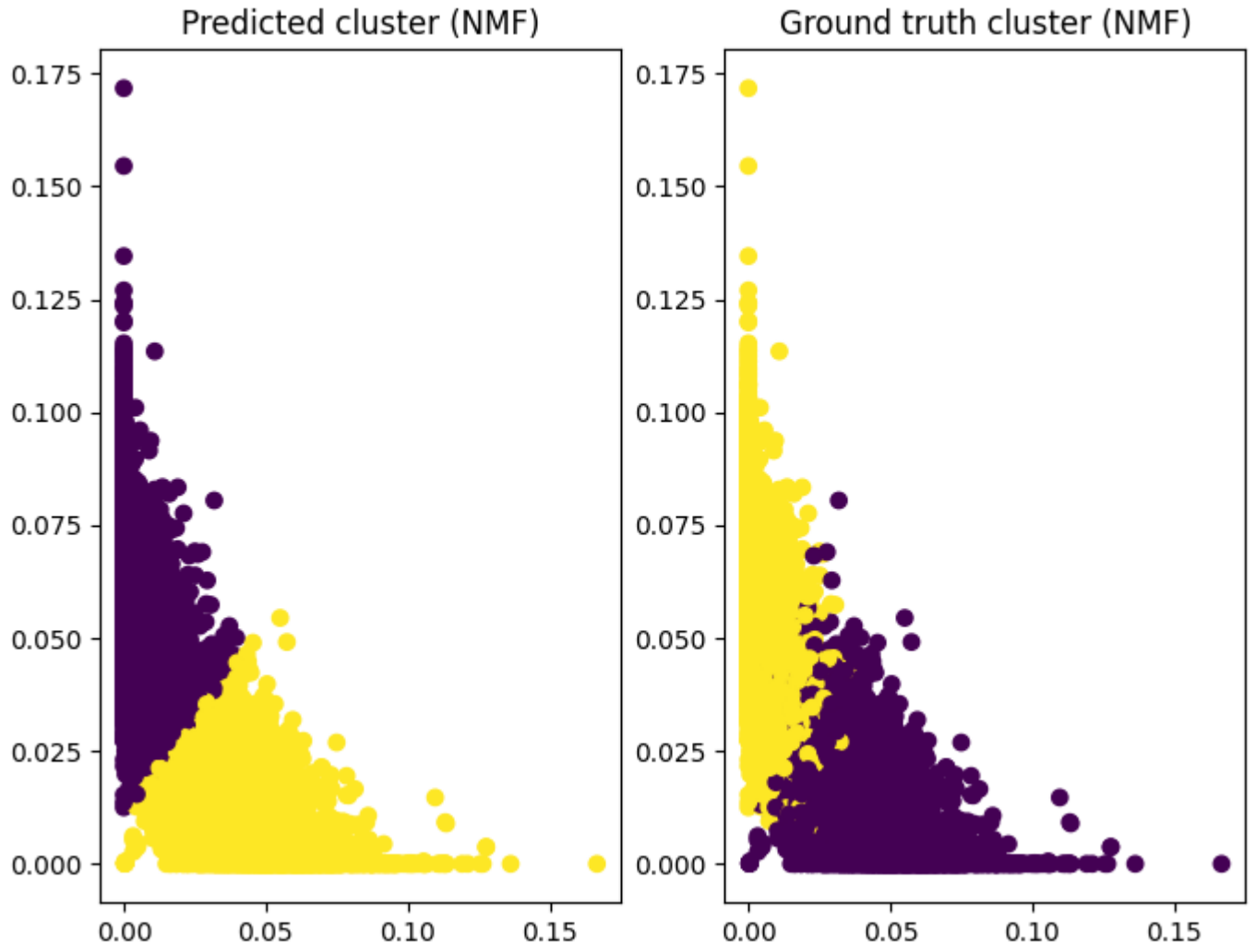
```
Homogeneity score : 0.595142438971607
Completeness_score score : 0.5959026162005286
V-measure score is: 0.595522284996832
Adjusted Rand Index score : 0.6981468460059148
Adjusted mutual information score : 0.5954852312955647
```



In [ ]:

```
km.fit(data_nmf_2)
plt.figure(figsize=(8, 6))
plt.subplot(121)
plt.scatter(data_nmf_2[:, 0], data_nmf_2[:, 1], c=km.labels_)
plt.title("Predicted cluster (NMF)")
plt.subplot(122)
plt.scatter(data_nmf_2[:, 0], data_nmf_2[:, 1], c=label)
plt.title("Ground truth cluster (NMF)")

print("Homogeneity score :",
      homogeneity_score(label, km.labels_))
print("Completeness score score :",
      completeness_score(label, km.labels_))
print("V-measure score :",
      v_measure_score(label, km.labels_))
print("Adjusted Rand Index score is :",
      adjusted_rand_score(label, km.labels_))
print("Adjusted Mutual information score is:",
      adjusted_mutual_info_score(label, km.labels_))
```

```
Homogeneity score : 0.6790483562300399
Completeness score score : 0.680131609210451
V-measure score : 0.6795895510492934
Adjusted Rand Index score is : 0.7770177788377391
Adjusted Mutual information score is: 0.6795601939855643
```



ANSWER-9

We observe anisitropic nature in the data distributions when we reduce dims to r=2 in both the SVD and NMF plots.

K-means will not work well in this distribution for reasons as follows:

- K-means clustering works best when the groups it's trying to identify are round and evenly sized. But, when we look at the SVD and NMF plots, the shapes aren't spherical; they're more like stretched-out blobs which shows that it is anisotropic gaussian and not symmetric variance in all dimensions. NMF plot even shows these blobs in different sizes.
- K-means++ is a way to start off with cluster centers far apart from one another for optimal clustering. But in the case with SVD and NMF plots, the blobs are too close together making it hard to tell them apart. That's why it's tough for K-means to group them correctly, which we can see in the low value of 5 scores for how well the clusters turn out. K-means also as we saisd assumes that each group is made up of data points that are spread out normally in all directions in a standard gaussian. But with SVD and NMF, it's clear that the spread isn't even. When the spread of points is not univariant like this, cluster centers can get thrown off by just a few noise data points.

Clustering with 20 classes

SVD

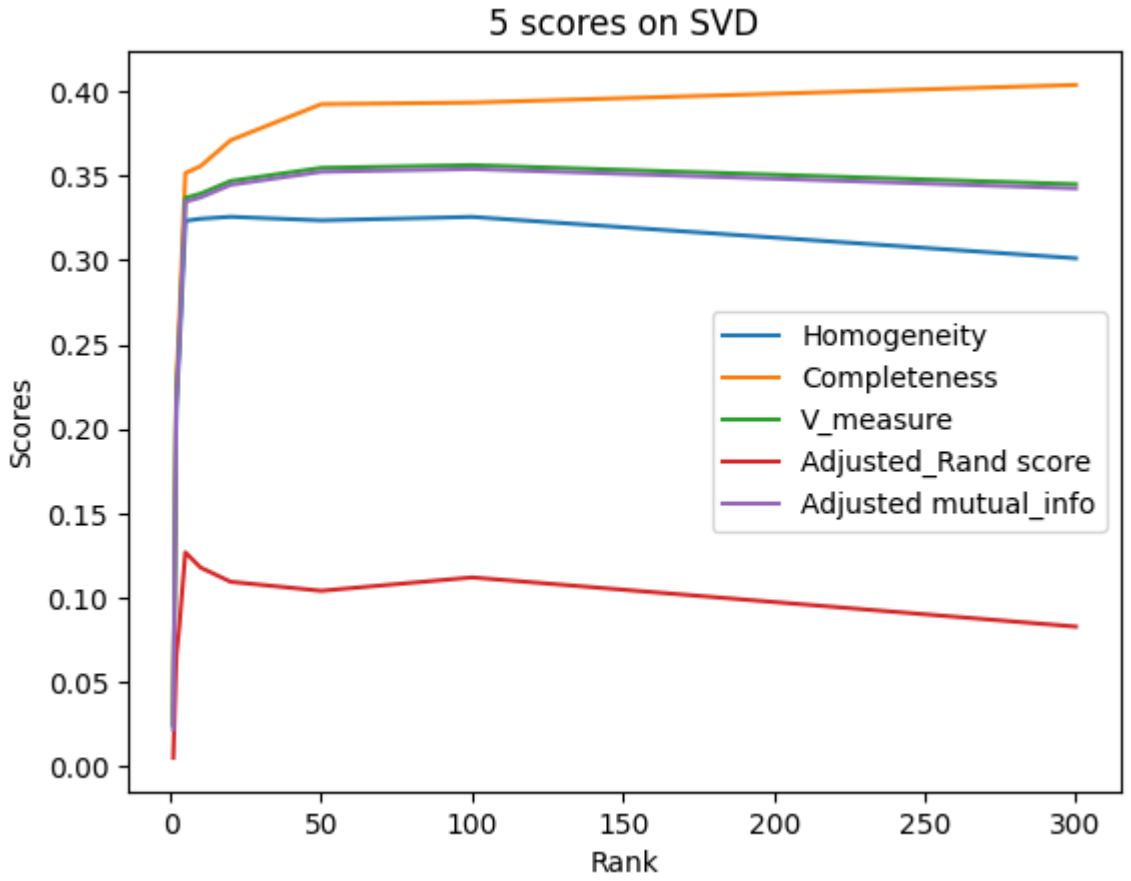
In [ ]:
```
dataset = fetch_20newsgroups(subset = 'all',shuffle = True, random_state = 42,remove=('heade
rs','footers'))
dvec_20 = CountVectorizer(stop_words='english',min_df=3)
tfidf_20 = TfidfTransformer()
data_dvec_20 = dvec_20.fit_transform(dataset.data)
tfidf_20 = tfidf_20.fit_transform(data_dvec_20)
```

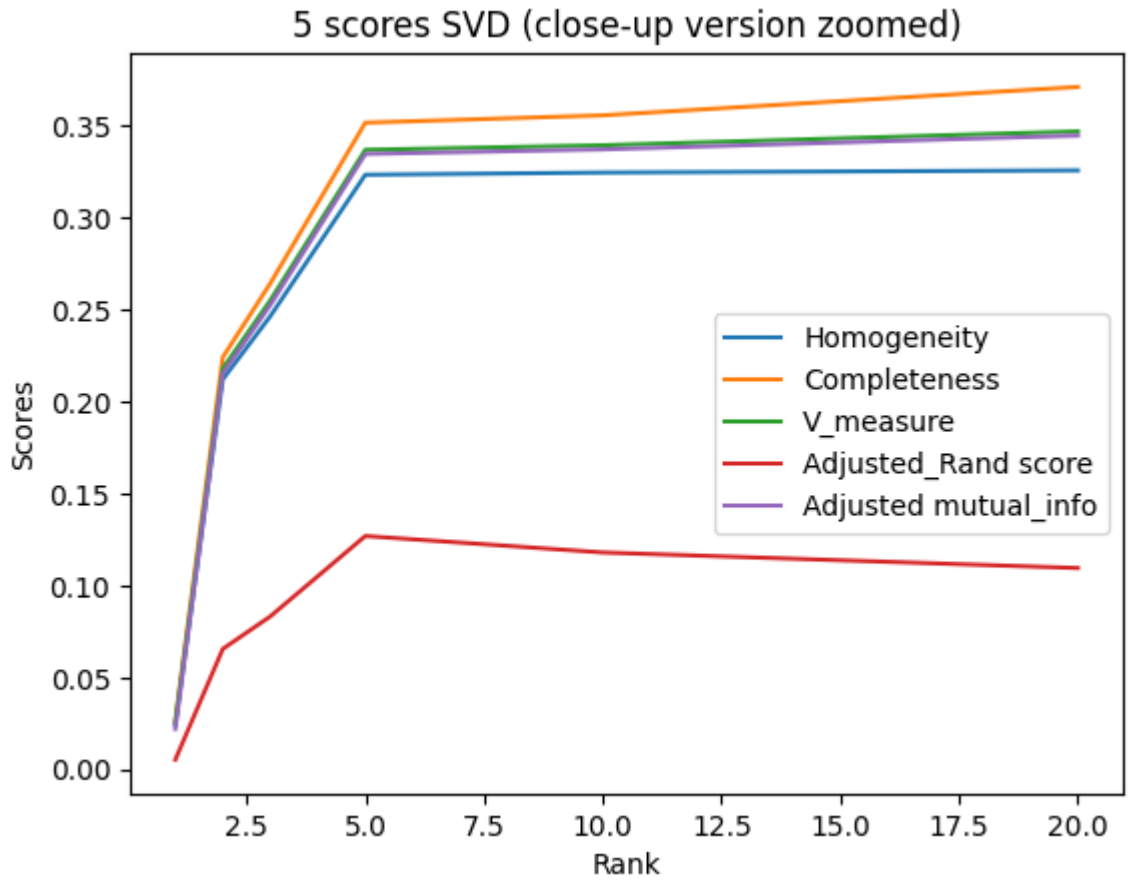
In [ ]:
```
h_s_svd = []
c_s_svd = []
v_s_svd = []
ar_s_svd = []
ami_s_svd = []

km_20 = KMeans(n_clusters=20, init='k-means++', max_iter=1500, n_init=30, random_state=42)
ranks = [1, 2, 3, 5, 10, 20, 50, 100, 300]

for rank in ranks:
    print("r=",rank)
    svd = TruncatedSVD(n_components=rank, random_state=42)
    svd_transformed_data = svd.fit_transform(tfidf_20)
    kmeans_svd = km_20.fit(svd_transformed_data)
    h_s_svd.append(homogeneity_score(dataset.target, kmeans_svd.labels_))
    c_s_svd.append(completeness_score(dataset.target, kmeans_svd.labels_))
    v_s_svd.append(v_measure_score(dataset.target, kmeans_svd.labels_))
    ar_s_svd.append(adjusted_rand_score(dataset.target, kmeans_svd.labels_))
    ami_s_svd.append(adjusted_mutual_info_score(dataset.target, kmeans_svd.labels_))
```

```
r= 1
r= 2
r= 3
r= 5
r= 10
r= 20
r= 50
r= 100
r= 300
```

In [ ]:

```
# plot 5 scores to find the best rank


plt.plot(ranks, h_s_svd)
plt.plot(ranks, c_s_svd)
plt.plot(ranks, v_s_svd)
plt.plot(ranks, ar_s_svd)
plt.plot(ranks, ami_s_svd)
plt.title('5 scores on SVD')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
plt.plot(ranks[:6], h_s_svd[:6])
plt.plot(ranks[:6], c_s_svd[:6])
plt.plot(ranks[:6], v_s_svd[:6])
plt.plot(ranks[:6], ar_s_svd[:6])
plt.plot(ranks[:6], ami_s_svd[:6])
plt.title('5 scores SVD (close-up version zoomed)')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
```


5 scores on SVD
Scores
Rank
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
50
100
150
200
250
300
Homogeneity
Completeness
V_measure
Adjusted_Rand score
Adjusted mutual_info



Best r = 5 for SVD

NMF

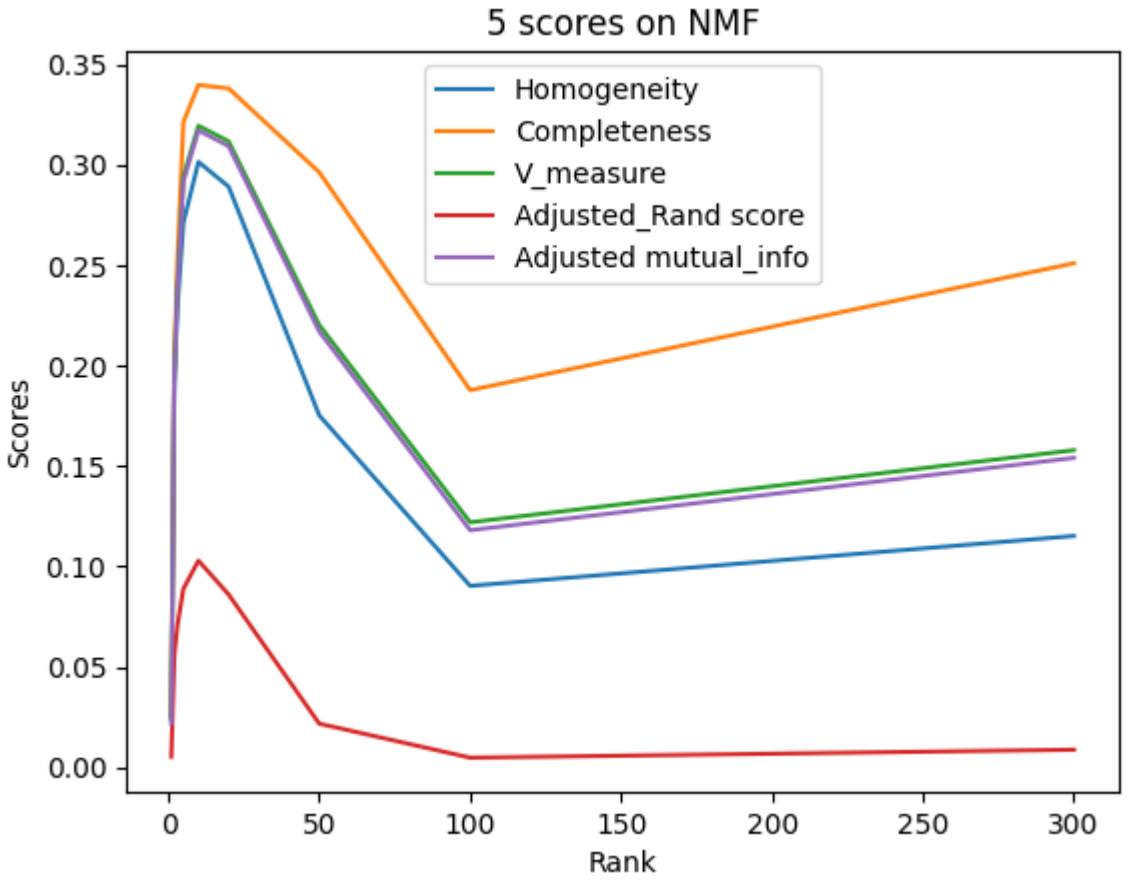
In [ ]:
```
h_s_nmf,c_s_nmf,v_s_nmf,ar_s_nmf,ami_s_nmf = [],[],[],[],[]
for rank in ranks:
    print("r=" , rank)
    nmf = NMF(n_components=rank)
    nmf_20 = nmf.fit_transform(tfidf_20)
    km_20.fit(nmf_20)
    h_s_nmf.append(homogeneity_score(dataset.target,km_20.labels_))
    c_s_nmf.append(completeness_score(dataset.target,km_20.labels_))
    v_s_nmf.append(v_measure_score(dataset.target,km_20.labels_))
    ar_s_nmf.append(adjusted_rand_score(dataset.target,km_20.labels_))
    ami_s_nmf.append(adjusted_mutual_info_score(dataset.target,km_20.labels_))
```

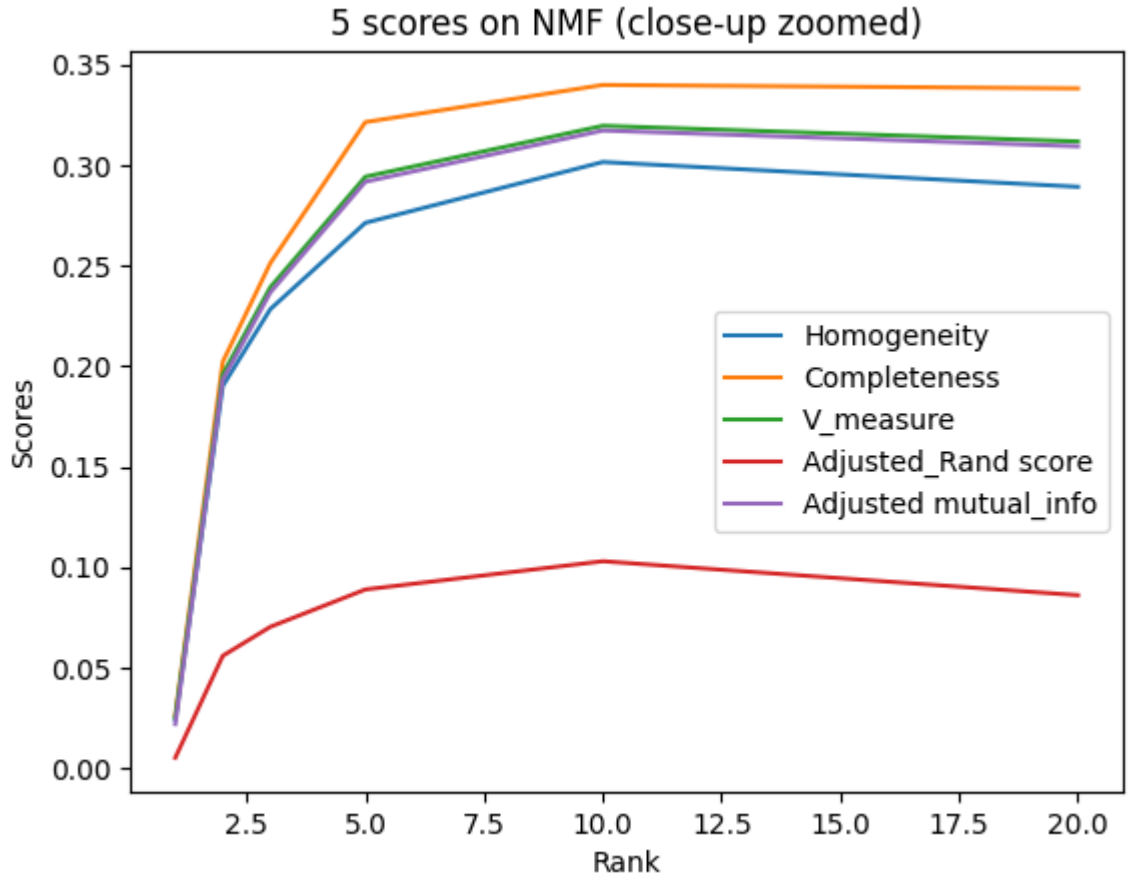
```
r= 1
r= 2
r= 3
r= 5
r= 10
r= 20
r= 50
r= 100
r= 300
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWarnin
g: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```

In [ ]:

```
# plot 5 scores to find the best rank
plt.plot(ranks, h_s_nmf)
plt.plot(ranks, c_s_nmf)
plt.plot(ranks, v_s_nmf)
plt.plot(ranks, ar_s_nmf)
plt.plot(ranks, ami_s_nmf)
plt.title('5 scores on NMF')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
# close-up plot
plt.plot(ranks[:6], h_s_nmf[:6])
plt.plot(ranks[:6], c_s_nmf[:6])
plt.plot(ranks[:6], v_s_nmf[:6])
plt.plot(ranks[:6], ar_s_nmf[:6])
plt.plot(ranks[:6], ami_s_nmf[:6])
plt.title('5 scores on NMF (close-up zoomed)')
plt.xlabel('Rank')
plt.ylabel('Scores')
plt.legend(labels = ['Homogeneity','Completeness',
                     'V_measure','Adjusted_Rand score','Adjusted mutual_info'])
plt.show()
```


5 scores on NMF
Homogeneity
Completeness
V_measure
Adjusted_Rand score
Adjusted mutual_info
Scores
Rank
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200
250
300



Best r = 10 for NMF

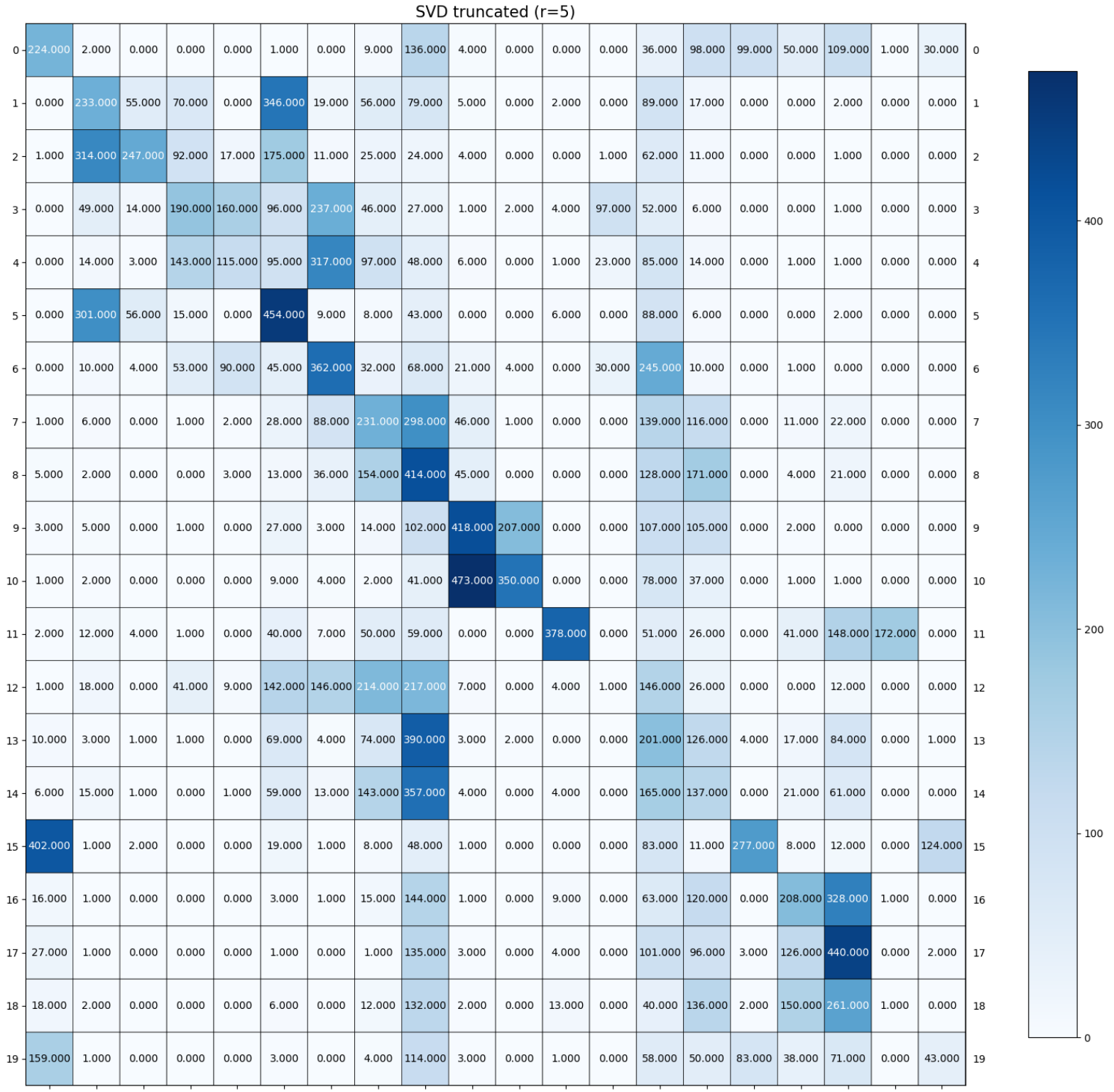
Truncate data for SVD and NMF

SVD

In [ ]:
```
r_svd, r_nmf = 5,10
svd_5 = TruncatedSVD(n_components=r_svd)
nmf_10 = NMF(n_components=r_nmf)
svd_5_ac = svd_5.fit_transform(tfidf_20)
nmf_10_ac = nmf_10.fit_transform(tfidf_20)
```

In [ ]:

```
from plotmat import plot_mat
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix
km_20_svd = KMeans(n_clusters=20, init='k-means++', max_iter=1000, n_init=30, random_state=4
2)

km_svd_all_r__5 = km_20_svd.fit(svd_5_ac)
cm_svd = confusion_matrix(dataset.target, km_svd_all_r__5.labels_)
rows, cols = linear_sum_assignment(cm_svd, maximize=True)
plot_mat(cm_svd[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,1
5),title = "SVD truncated (r=5)")
```

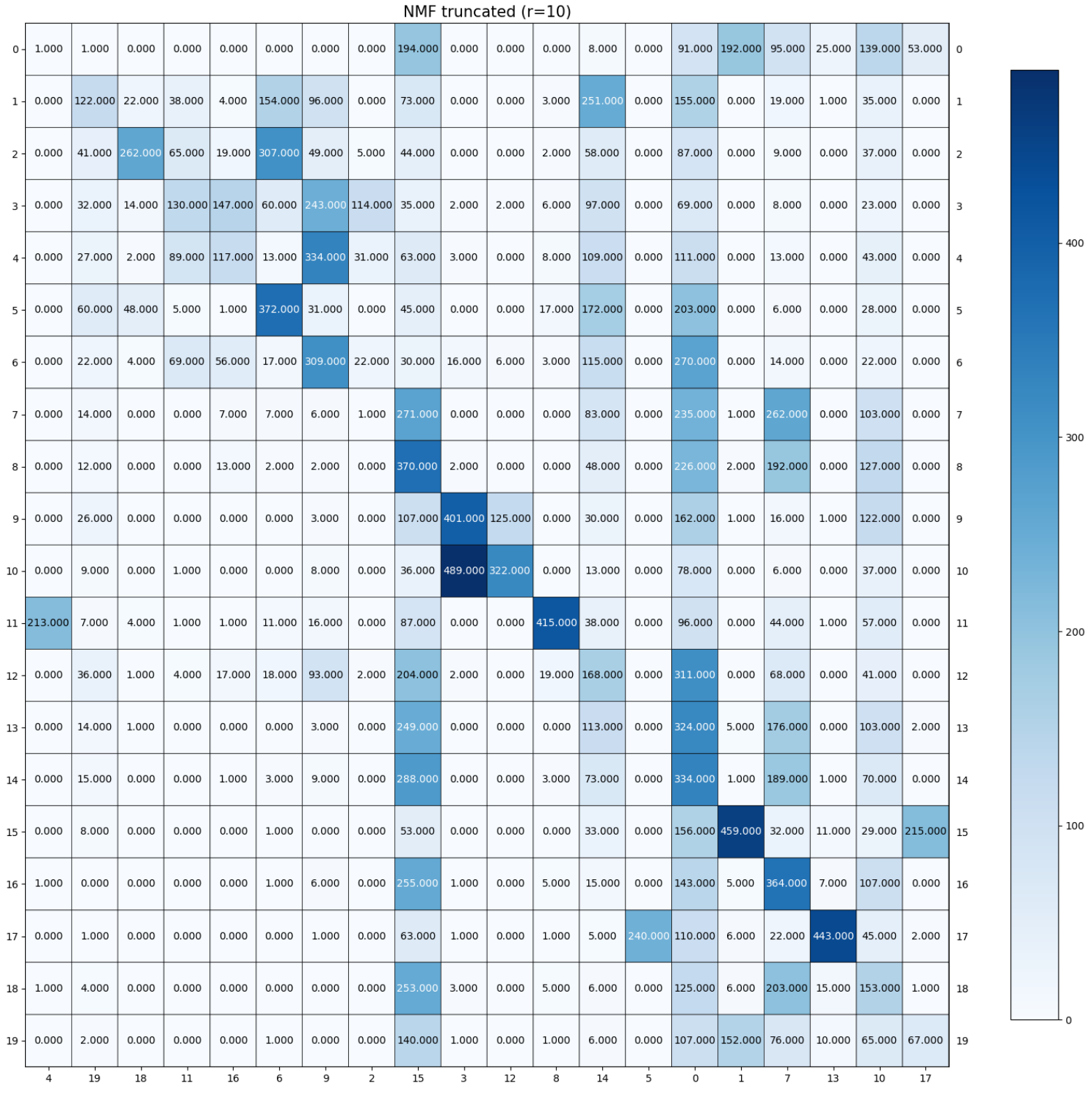

SVD truncated (r=5)
0 224.000 2.000 0.000 0.000 0.000 1.000 0.000 9.000 136.000 4.000 0.000 0.000 0.000 36.000 98.000 99.000 50.000 109.000 1.000 30.000 0
1 0.000 233.000 55.000 70.000 0.000 346.000 19.000 56.000 79.000 5.000 0.000 2.000 0.000 89.000 17.000 0.000 0.000 2.000 0.000 0.000 1
2 1.000 314.000 247.000 92.000 17.000 175.000 11.000 25.000 24.000 4.000 0.000 0.000 1.000 62.000 11.000 0.000 0.000 1.000 0.000 0.000 2
3 0.000 49.000 14.000 190.000 160.000 96.000 237.000 46.000 27.000 1.000 2.000 4.000 97.000 52.000 6.000 0.000 0.000 1.000 0.000 0.000 3
4 0.000 14.000 3.000 143.000 115.000 95.000 317.000 97.000 48.000 6.000 0.000 1.000 23.000 85.000 14.000 0.000 1.000 1.000 0.000 0.000 4
5 0.000 301.000 56.000 15.000 0.000 454.000 9.000 8.000 43.000 0.000 0.000 6.000 0.000 88.000 6.000 0.000 0.000 2.000 0.000 0.000 5
6 0.000 10.000 4.000 53.000 90.000 45.000 362.000 32.000 68.000 21.000 4.000 0.000 30.000 245.000 10.000 0.000 1.000 0.000 0.000 0.000 6
7 1.000 6.000 0.000 1.000 2.000 28.000 88.000 231.000 298.000 46.000 1.000 0.000 0.000 139.000 116.000 0.000 11.000 22.000 0.000 0.000 7
8 5.000 2.000 0.000 0.000 3.000 13.000 36.000 154.000 414.000 45.000 0.000 0.000 0.000 128.000 171.000 0.000 4.000 21.000 0.000 0.000 8
9 3.000 5.000 0.000 1.000 0.000 27.000 3.000 14.000 102.000 418.000 207.000 0.000 0.000 107.000 105.000 0.000 2.000 0.000 0.000 0.000 9
10 1.000 2.000 0.000 0.000 0.000 9.000 4.000 2.000 41.000 473.000 350.000 0.000 0.000 78.000 37.000 0.000 1.000 1.000 0.000 0.000 10
11 2.000 12.000 4.000 1.000 0.000 40.000 7.000 50.000 59.000 0.000 0.000 378.000 0.000 51.000 26.000 0.000 41.000 148.000 172.000 0.000 11
12 1.000 18.000 0.000 41.000 9.000 142.000 146.000 214.000 217.000 7.000 0.000 4.000 1.000 146.000 26.000 0.000 0.000 12.000 0.000 0.000 12
13 10.000 3.000 1.000 1.000 0.000 69.000 4.000 74.000 390.000 3.000 2.000 0.000 0.000 201.000 126.000 4.000 17.000 84.000 0.000 1.000 13
14 6.000 15.000 1.000 0.000 1.000 59.000 13.000 143.000 357.000 4.000 0.000 4.000 0.000 165.000 137.000 0.000 21.000 61.000 0.000 0.000 14
15 402.000 1.000 2.000 0.000 0.000 19.000 1.000 8.000 48.000 1.000 0.000 0.000 0.000 83.000 11.000 277.000 8.000 12.000 0.000 124.000 15
16 16.000 1.000 0.000 0.000 0.000 3.000 1.000 15.000 144.000 1.000 0.000 9.000 0.000 63.000 120.000 0.000 208.000 328.000 1.000 0.000 16
17 27.000 1.000 0.000 0.000 0.000 1.000 0.000 1.000 135.000 3.000 0.000 4.000 0.000 101.000 96.000 3.000 126.000 440.000 0.000 2.000 17
18 18.000 2.000 0.000 0.000 0.000 6.000 0.000 12.000 132.000 2.000 0.000 13.000 0.000 40.000 136.000 2.000 150.000 261.000 1.000 0.000 18
19 159.000 1.000 0.000 0.000 0.000 3.000 0.000 4.000 114.000 3.000 0.000 1.000 0.000 58.000 50.000 83.000 38.000 71.000 0.000 43.000 19
6 7 5 4 9 15 17 2 16 19 0 14 13 1 10 11 8 18 12 3
400
300
200
100
0

In [ ]:
```
print("############# SVD Truncated Scores ####################")
print("Homogeneity score :",
      homogeneity_score(dataset.target, km_svd_all_r__5.labels_))
print("Completeness score score :",
      completeness_score(dataset.target, km_svd_all_r__5.labels_))
print("V-measure score :",
      v_measure_score(dataset.target, km_svd_all_r__5.labels_))
print("Adjusted Rand Index score is :",
      adjusted_rand_score(dataset.target, km_svd_all_r__5.labels_))
print("Adjusted Mutual information score is:",
      adjusted_mutual_info_score(dataset.target, km_svd_all_r__5.labels_))
```

```
############# SVD Truncated Scores ####################
Homogeneity score : 0.3234034902259969
Completeness score score : 0.35162912368425425
V-measure score : 0.33692619740508106
Adjusted Rand Index score is : 0.12688298800519768
Adjusted Mutual information score is: 0.3346883593503683
```

NMF

In [ ]:
```
km_20_nmf = KMeans(n_clusters=20, init='k-means++', max_iter=1000, n_init=30, random_state=4
2)
km_nmf_all_r__10 = km_20_nmf.fit(nmf_10_ac)
cm_nmf = confusion_matrix(dataset.target, km_nmf_all_r__10.labels_)
rows, cols = linear_sum_assignment(cm_nmf, maximize=True)
plot_mat(cm_nmf[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,1
5),title = "NMF truncated (r=10)")
```


NMF truncated (r=10)
0 1.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 194.000 0.000 0.000 0.000 8.000 0.000 91.000 192.000 95.000 25.000 139.000 53.000 0
1 0.000 122.000 22.000 38.000 4.000 154.000 96.000 0.000 73.000 0.000 0.000 3.000 251.000 0.000 155.000 0.000 19.000 1.000 35.000 0.000 1
2 0.000 41.000 262.000 65.000 19.000 307.000 49.000 5.000 44.000 0.000 0.000 2.000 58.000 0.000 87.000 0.000 9.000 0.000 37.000 0.000 2
3 0.000 32.000 14.000 130.000 147.000 60.000 243.000 114.000 35.000 2.000 2.000 6.000 97.000 0.000 69.000 0.000 8.000 0.000 23.000 0.000 3
4 0.000 27.000 2.000 89.000 117.000 13.000 334.000 31.000 63.000 3.000 0.000 8.000 109.000 0.000 111.000 0.000 13.000 0.000 43.000 0.000 4
5 0.000 60.000 48.000 5.000 1.000 372.000 31.000 0.000 45.000 0.000 0.000 17.000 172.000 0.000 203.000 0.000 6.000 0.000 28.000 0.000 5
6 0.000 22.000 4.000 69.000 56.000 17.000 309.000 22.000 30.000 16.000 6.000 3.000 115.000 0.000 270.000 0.000 14.000 0.000 22.000 0.000 6
7 0.000 14.000 0.000 0.000 7.000 7.000 6.000 1.000 271.000 0.000 0.000 0.000 83.000 0.000 235.000 1.000 262.000 0.000 103.000 0.000 7
8 0.000 12.000 0.000 0.000 13.000 2.000 2.000 0.000 370.000 2.000 0.000 0.000 48.000 0.000 226.000 2.000 192.000 0.000 127.000 0.000 8
9 0.000 26.000 0.000 0.000 0.000 0.000 3.000 0.000 107.000 401.000 125.000 0.000 30.000 0.000 162.000 1.000 16.000 1.000 122.000 0.000 9
10 0.000 9.000 0.000 1.000 0.000 0.000 8.000 0.000 36.000 489.000 322.000 0.000 13.000 0.000 78.000 0.000 6.000 0.000 37.000 0.000 10
11 213.000 7.000 4.000 1.000 1.000 11.000 16.000 0.000 87.000 0.000 0.000 415.000 38.000 0.000 96.000 0.000 44.000 1.000 57.000 0.000 11
12 0.000 36.000 1.000 4.000 17.000 18.000 93.000 2.000 204.000 2.000 0.000 19.000 168.000 0.000 311.000 0.000 68.000 0.000 41.000 0.000 12
13 0.000 14.000 1.000 0.000 0.000 0.000 3.000 0.000 249.000 0.000 0.000 0.000 113.000 0.000 324.000 5.000 176.000 0.000 103.000 2.000 13
14 0.000 15.000 0.000 0.000 1.000 3.000 9.000 0.000 288.000 0.000 0.000 3.000 73.000 0.000 334.000 1.000 189.000 1.000 70.000 0.000 14
15 0.000 8.000 0.000 0.000 0.000 1.000 0.000 0.000 53.000 0.000 0.000 0.000 33.000 0.000 156.000 459.000 32.000 11.000 29.000 215.000 15
16 1.000 0.000 0.000 0.000 0.000 1.000 6.000 0.000 255.000 1.000 0.000 5.000 15.000 0.000 143.000 5.000 364.000 7.000 107.000 0.000 16
17 0.000 1.000 0.000 0.000 0.000 0.000 1.000 0.000 63.000 1.000 0.000 1.000 5.000 240.000 110.000 6.000 22.000 443.000 45.000 2.000 17
18 1.000 4.000 0.000 0.000 0.000 0.000 0.000 0.000 253.000 3.000 0.000 5.000 6.000 0.000 125.000 6.000 203.000 15.000 153.000 1.000 18
19 0.000 2.000 0.000 0.000 0.000 1.000 0.000 0.000 140.000 1.000 0.000 1.000 6.000 0.000 107.000 152.000 76.000 10.000 65.000 67.000 19
4 19 18 11 16 6 9 2 15 3 12 8 14 5 0 1 7 13 10 17
400
300
200
100
0

In [ ]:
```
print("############# NMF Truncated Scores ####################")
print("Homogeneity score :",
      homogeneity_score(dataset.target, km_nmf_all_r__10.labels_))
print("Completeness score score :",
      completeness_score(dataset.target, km_nmf_all_r__10.labels_))
print("V-measure score :",
      v_measure_score(dataset.target, km_nmf_all_r__10.labels_))
print("Adjusted Rand Index score is :",
      adjusted_rand_score(dataset.target, km_nmf_all_r__10.labels_))
print("Adjusted Mutual information score is:",
      adjusted_mutual_info_score(dataset.target, km_nmf_all_r__10.labels_))
```

```
############# NMF Truncated Scores ####################
Homogeneity score : 0.3013011867132605
Completeness score score : 0.33947726117776017
V-measure score : 0.31925200353312305
Adjusted Rand Index score is : 0.10294726949206563
Adjusted Mutual information score is: 0.316914981488837
```

Answer-10

First, we chose best 'r' for SVD and NMF respectively, and now we run K-Means on them to obtain below contingency matrices as shown in the plots above. 5 metric scores for each SVM and NMF with best 'r' are shown below :

	Homogeneity Score	Completeness Score	V-Measure Score	Adjusted Rand Index	Adjusted Mutual Information Score
SVD Truncated Scores	0.323	0.352	0.337	0.127	0.335
NMF Truncated Scores	0.301	0.339	0.319	0.103	0.317

UMAP

In [ ]:

```
h_s_euc = []
c_s_euc = []
v_s_euc = []
ar_s_euc = []
ami_s_euc = []
h_s_cos = []
c_s_cos = []
v_s_cos = []
ar_s_cos = []
ami_s_cos = []
ranks = [5,20,200]
km_20 = KMeans(n_clusters=20, init='k-means++', max_iter=1500, n_init=30, random_state=42)
for rank in ranks:
    print(' UMAP Euclidean  r = ', rank)
    umap_euc = umap.UMAP(n_components=rank, metric='euclidean').fit_transform(tfidf_20)
    kmean_euc = km_20.fit(umap_euc)
    h_s_euc.append(homogeneity_score(dataset.target, kmean_euc.labels_))
    c_s_euc.append(completeness_score(dataset.target, kmean_euc.labels_))
    v_s_euc.append(v_measure_score(dataset.target, kmean_euc.labels_))
    ar_s_euc.append(adjusted_rand_score(dataset.target, kmean_euc.labels_))
    ami_s_euc.append(adjusted_mutual_info_score(dataset.target, kmean_euc.labels_))

    print(' UMAP Cosine, r = ', rank)
    umap_cos = umap.UMAP(n_components=rank, metric='cosine').fit_transform(tfidf_20)
    kmean_cos = km_20.fit(umap_cos)
    h_s_cos.append(homogeneity_score(dataset.target, kmean_cos.labels_))
    c_s_euc.append(completeness_score(dataset.target, kmean_cos.labels_))
    v_s_euc.append(v_measure_score(dataset.target, kmean_cos.labels_))
    ar_s_euc.append(adjusted_rand_score(dataset.target, kmean_cos.labels_))
    ami_s_euc.append(adjusted_mutual_info_score(dataset.target, kmean_cos.labels_))
```

```
UMAP Euclidean  r =  5
UMAP Cosine, r =  5
UMAP Euclidean  r =  20
UMAP Cosine, r =  20
UMAP Euclidean  r =  200
UMAP Cosine, r =  200
```

In [ ]:

```
label = dataset.target

print("-----------------  UMAP (Euclidean)  -----------------\n")
for i in range(len(ranks)):
    print("\nRank={} ".format(ranks[i]))
    print("***********\n")
    print("Homogeneity score :", h_s_euc[i])
    print("Completeness score :", c_s_euc[2*i])
    print("V-measure score :", v_s_euc[2*i])
    print("Adjusted Rand Index score :", ar_s_euc[2*i])
    print("Adjusted Mutual information score :", ami_s_euc[2*i])

print("\n-----------------  UMAP (Cosine)  -----------------\n")
for i in range(len(ranks)):
    print("\nRank={}".format(ranks[i]))
    print("***********\n")
    print("Homogeneity score :", h_s_cos[i])
    print("Completeness score :", c_s_euc[2*i+1])
    print("V-measure score :", v_s_euc[2*i+1])
    print("Adjusted Rand Index score :", ar_s_euc[2*i+1])
    print("Adjusted Mutual information score :", ami_s_euc[2*i+1])
```

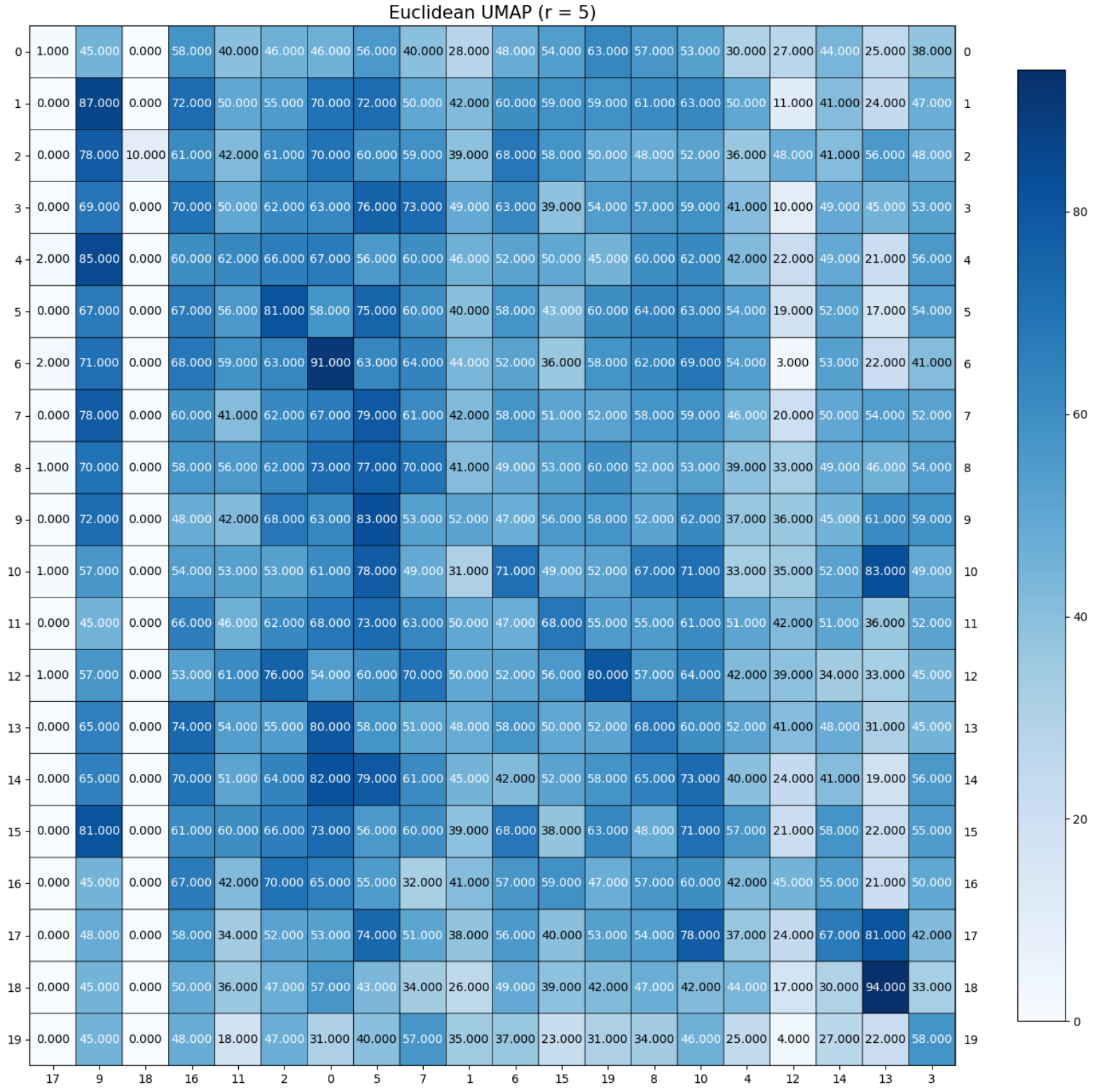
```
-----------------  UMAP (Euclidean)  -----------------


Rank=5
***********

Homogeneity score : 0.006841303977673622
Completeness score : 0.007091416567500562
V-measure score : 0.006964115330280703
Adjusted Rand Index score : 0.0009578016720213048
Adjusted Mutual information score : 0.003706469184580692

Rank=20
***********

Homogeneity score : 0.00759271224822598
Completeness score : 0.007730153593475889
V-measure score : 0.007660816517771405
Adjusted Rand Index score : 0.0009746129734326004
Adjusted Mutual information score : 0.004426438274208731

Rank=200
***********

Homogeneity score : 0.006846613057672704
Completeness score : 0.007108161411607371
V-measure score : 0.006974936190318378
Adjusted Rand Index score : 0.0007528907812050753
Adjusted Mutual information score : 0.0037314028276295313

-----------------  UMAP (Cosine)  -----------------


Rank=5
***********

Homogeneity score : 0.5681500291547111
Completeness score : 0.5945985755438022
V-measure score : 0.5810734954494379
Adjusted Rand Index score : 0.44334860688635175
Adjusted Mutual information score : 0.5796774224598082

Rank=20
***********

Homogeneity score : 0.5656288528891319
```

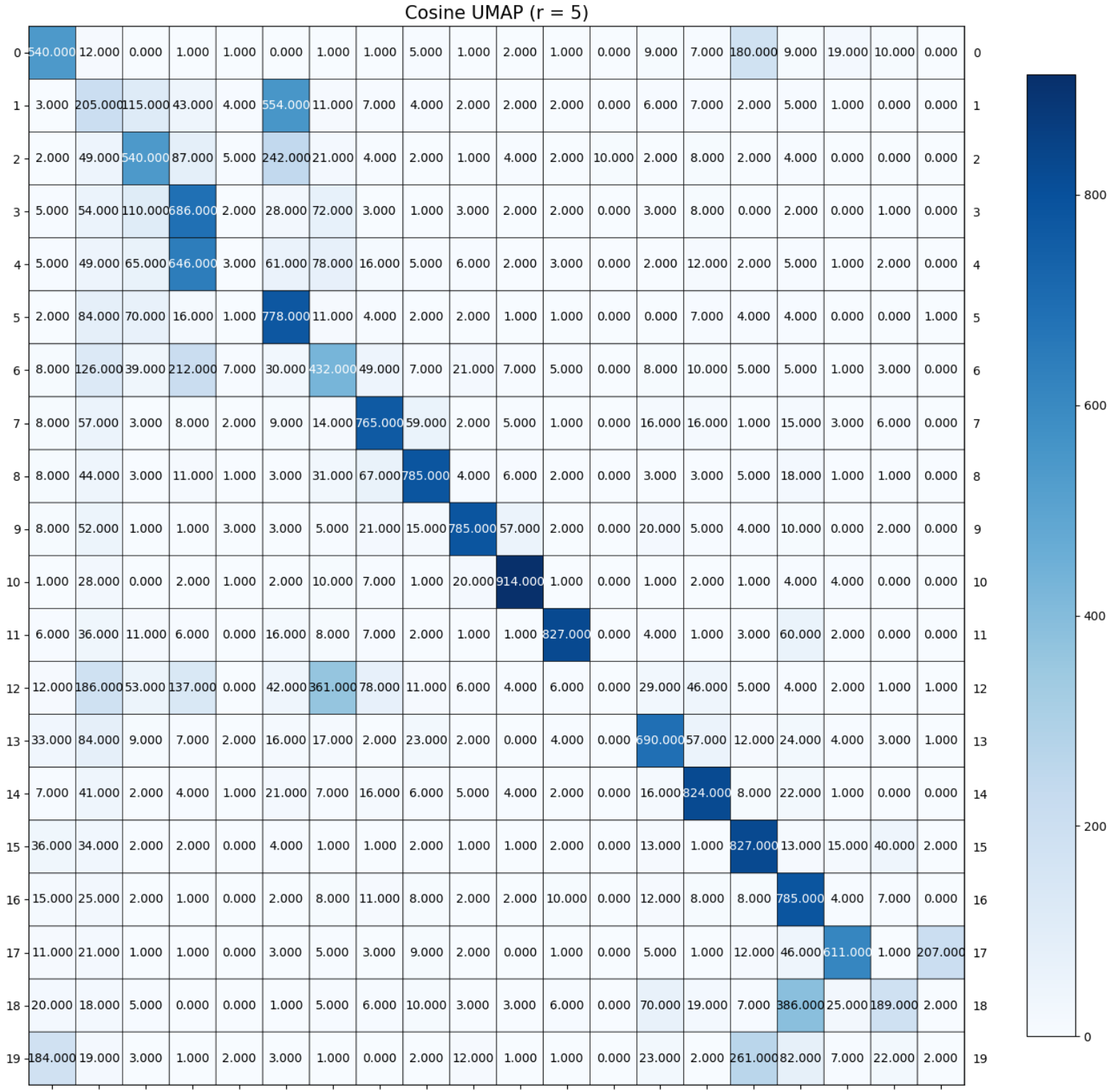
```
Completeness score : 0.5923234721431395
V-measure score : 0.5786684630186212
Adjusted Rand Index score : 0.44690454022951814
Adjusted Mutual information score : 0.5772646280558906

Rank=200
***********

Homogeneity score : 0.5657026609680136
Completeness score : 0.5915689953759588
V-measure score : 0.5783467572127019
Adjusted Rand Index score : 0.4401495459445356
Adjusted Mutual information score : 0.5769431257794201
```

In [ ]:

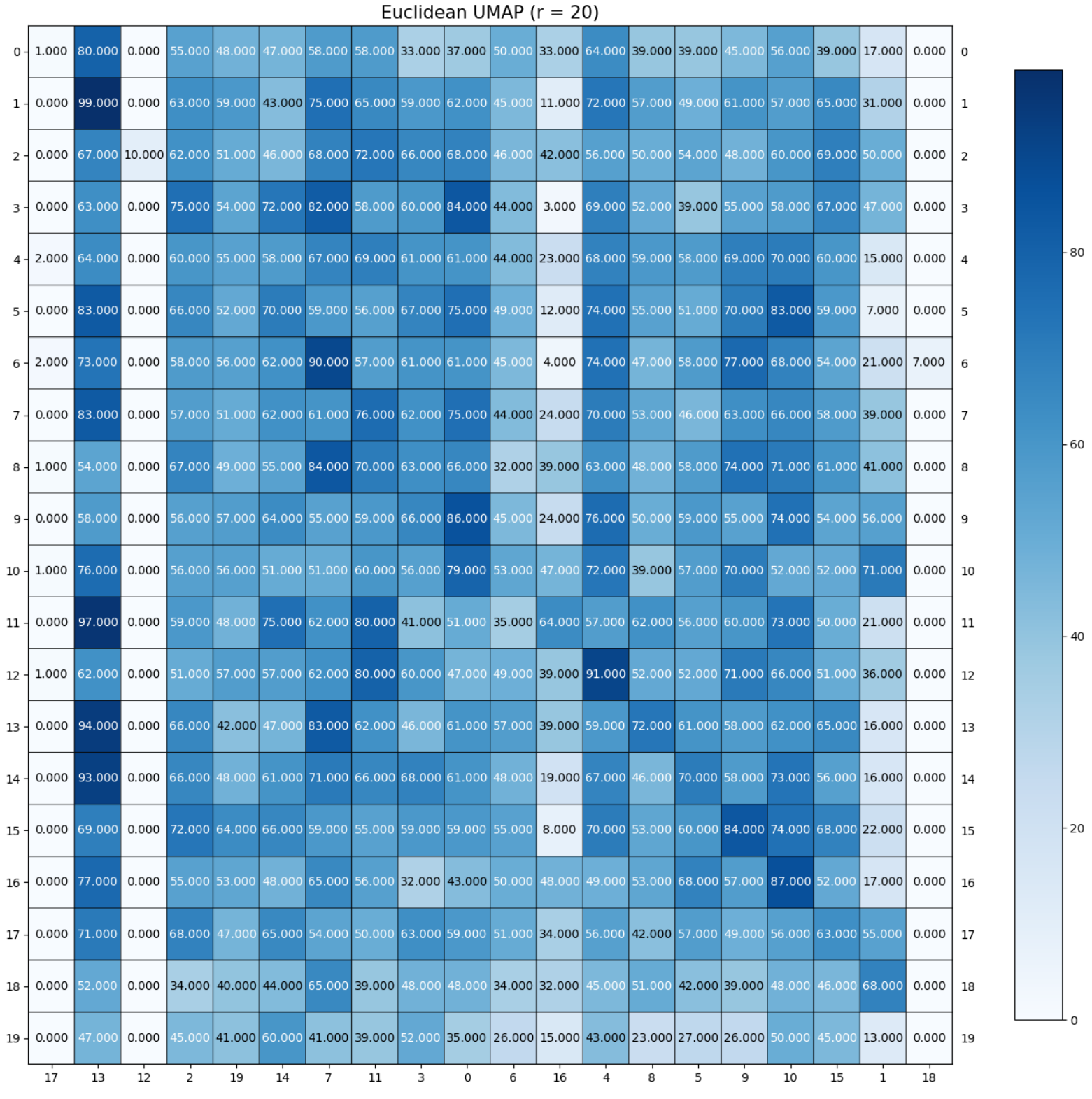
```
ranks = [5,20,200]
km_20 = KMeans(n_clusters=20, init='k-means++', max_iter=1500, n_init=30, random_state=42)
label  = dataset.target
for rank in ranks:
    print("Rank : ", rank)
    umap_euc = umap.UMAP(n_components=rank, metric='euclidean').fit_transform(tfidf_20)
    kmean_euc = km_20.fit(umap_euc)
    cm = confusion_matrix(label, kmean_euc.labels_)
    rows, cols = linear_sum_assignment(cm, maximize=True)
    plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, title = 'Euc
lidean UMAP (r = {})'.format(rank), size=(13,13))

    umap_cos = umap.UMAP(n_components=rank, metric='cosine').fit_transform(tfidf_20)
    kmean_cos = km_20.fit(umap_cos)
    cm = confusion_matrix(label, kmean_cos.labels_)
    rows, cols = linear_sum_assignment(cm, maximize=True)
    plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, title = 'Cos
ine UMAP (r = {})'.format(rank), size=(13,13))
```

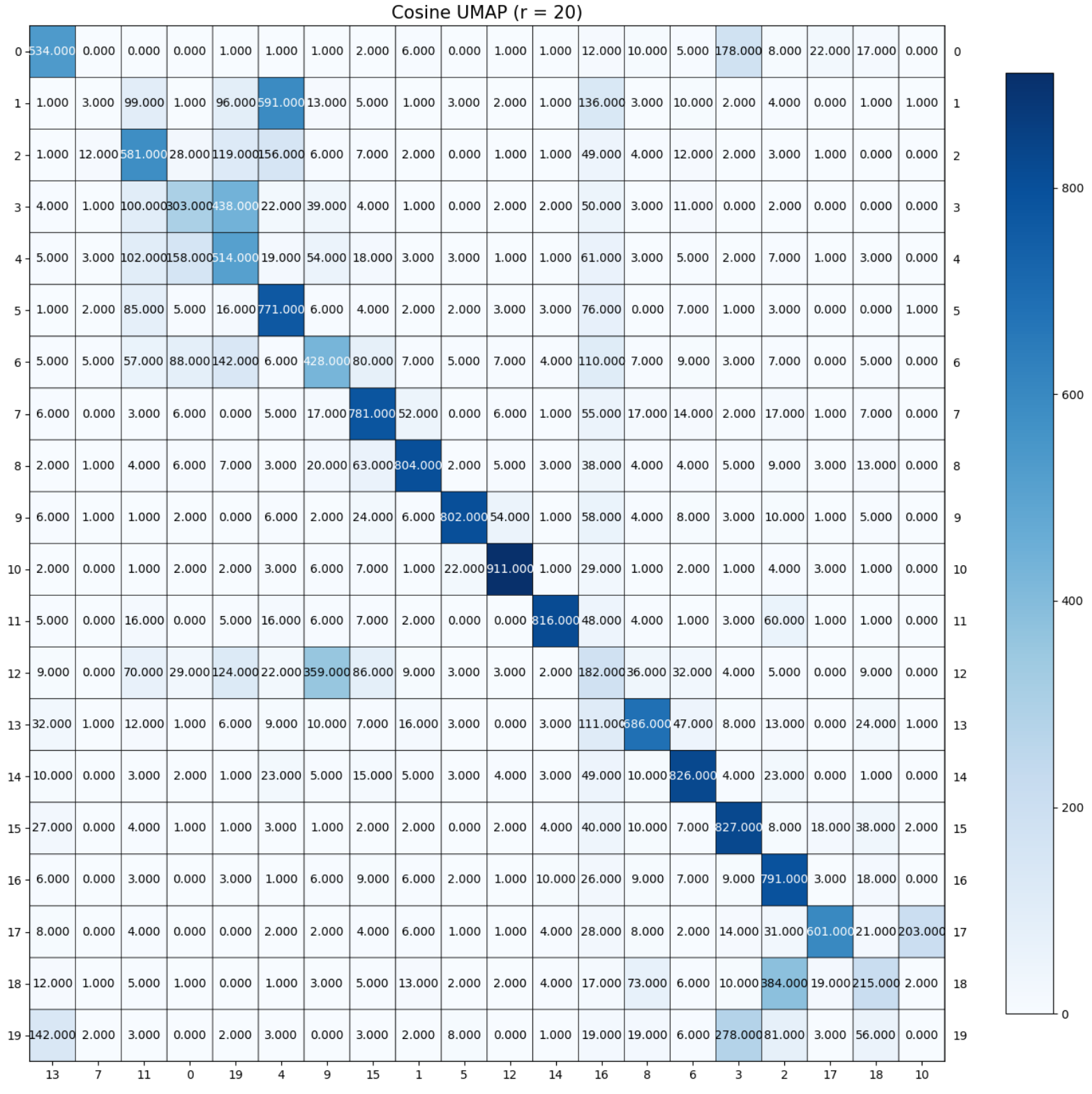
```
Rank :  5
```


Euclidean UMAP (r = 5)
0 1.000 45.000 0.000 58.000 40.000 46.000 46.000 56.000 40.000 28.000 48.000 54.000 63.000 57.000 53.000 30.000 27.000 44.000 25.000 38.000 0
1 0.000 87.000 0.000 72.000 50.000 55.000 70.000 72.000 50.000 42.000 60.000 59.000 59.000 61.000 63.000 50.000 11.000 41.000 24.000 47.000 1
2 0.000 78.000 10.000 61.000 42.000 61.000 70.000 60.000 59.000 39.000 68.000 58.000 50.000 48.000 52.000 36.000 48.000 41.000 56.000 48.000 2
3 0.000 69.000 0.000 70.000 50.000 62.000 63.000 76.000 73.000 49.000 63.000 39.000 54.000 57.000 59.000 41.000 10.000 49.000 45.000 53.000 3
4 2.000 85.000 0.000 60.000 62.000 66.000 67.000 56.000 60.000 46.000 52.000 50.000 45.000 60.000 62.000 42.000 22.000 49.000 21.000 56.000 4
5 0.000 67.000 0.000 67.000 56.000 81.000 58.000 75.000 60.000 40.000 58.000 43.000 60.000 64.000 63.000 54.000 19.000 52.000 17.000 54.000 5
6 2.000 71.000 0.000 68.000 59.000 63.000 91.000 63.000 64.000 44.000 52.000 36.000 58.000 62.000 69.000 54.000 3.000 53.000 22.000 41.000 6
7 0.000 78.000 0.000 60.000 41.000 62.000 67.000 79.000 61.000 42.000 58.000 51.000 52.000 58.000 59.000 46.000 20.000 50.000 54.000 52.000 7
8 1.000 70.000 0.000 58.000 56.000 62.000 73.000 77.000 70.000 41.000 49.000 53.000 60.000 52.000 53.000 39.000 33.000 49.000 46.000 54.000 8
9 0.000 72.000 0.000 48.000 42.000 68.000 63.000 83.000 53.000 52.000 47.000 56.000 58.000 52.000 62.000 37.000 36.000 45.000 61.000 59.000 9
10 1.000 57.000 0.000 54.000 53.000 53.000 61.000 78.000 49.000 31.000 71.000 49.000 52.000 67.000 71.000 33.000 35.000 52.000 83.000 49.000 10
11 0.000 45.000 0.000 66.000 46.000 62.000 68.000 73.000 63.000 50.000 47.000 68.000 55.000 55.000 61.000 51.000 42.000 51.000 36.000 52.000 11
12 1.000 57.000 0.000 53.000 61.000 76.000 54.000 60.000 70.000 50.000 52.000 56.000 80.000 57.000 64.000 42.000 39.000 34.000 33.000 45.000 12
13 0.000 65.000 0.000 74.000 54.000 55.000 80.000 58.000 51.000 48.000 58.000 50.000 52.000 68.000 60.000 52.000 41.000 48.000 31.000 45.000 13
14 0.000 65.000 0.000 70.000 51.000 64.000 82.000 79.000 61.000 45.000 42.000 52.000 58.000 65.000 73.000 40.000 24.000 41.000 19.000 56.000 14
15 0.000 81.000 0.000 61.000 60.000 66.000 73.000 56.000 60.000 39.000 68.000 38.000 63.000 48.000 71.000 57.000 21.000 58.000 22.000 55.000 15
16 0.000 45.000 0.000 67.000 42.000 70.000 65.000 55.000 32.000 41.000 57.000 59.000 47.000 57.000 60.000 42.000 45.000 55.000 21.000 50.000 16
17 0.000 48.000 0.000 58.000 34.000 52.000 53.000 74.000 51.000 38.000 56.000 40.000 53.000 54.000 78.000 37.000 24.000 67.000 81.000 42.000 17
18 0.000 45.000 0.000 50.000 36.000 47.000 57.000 43.000 34.000 26.000 49.000 39.000 42.000 47.000 42.000 44.000 17.000 30.000 94.000 33.000 18
19 0.000 45.000 0.000 48.000 18.000 47.000 31.000 40.000 57.000 35.000 37.000 23.000 31.000 34.000 46.000 25.000 4.000 27.000 22.000 58.000 19
17 9 18 16 11 2 0 5 7 1 6 15 19 8 10 4 12 14 13 3
80
60
40
20
0

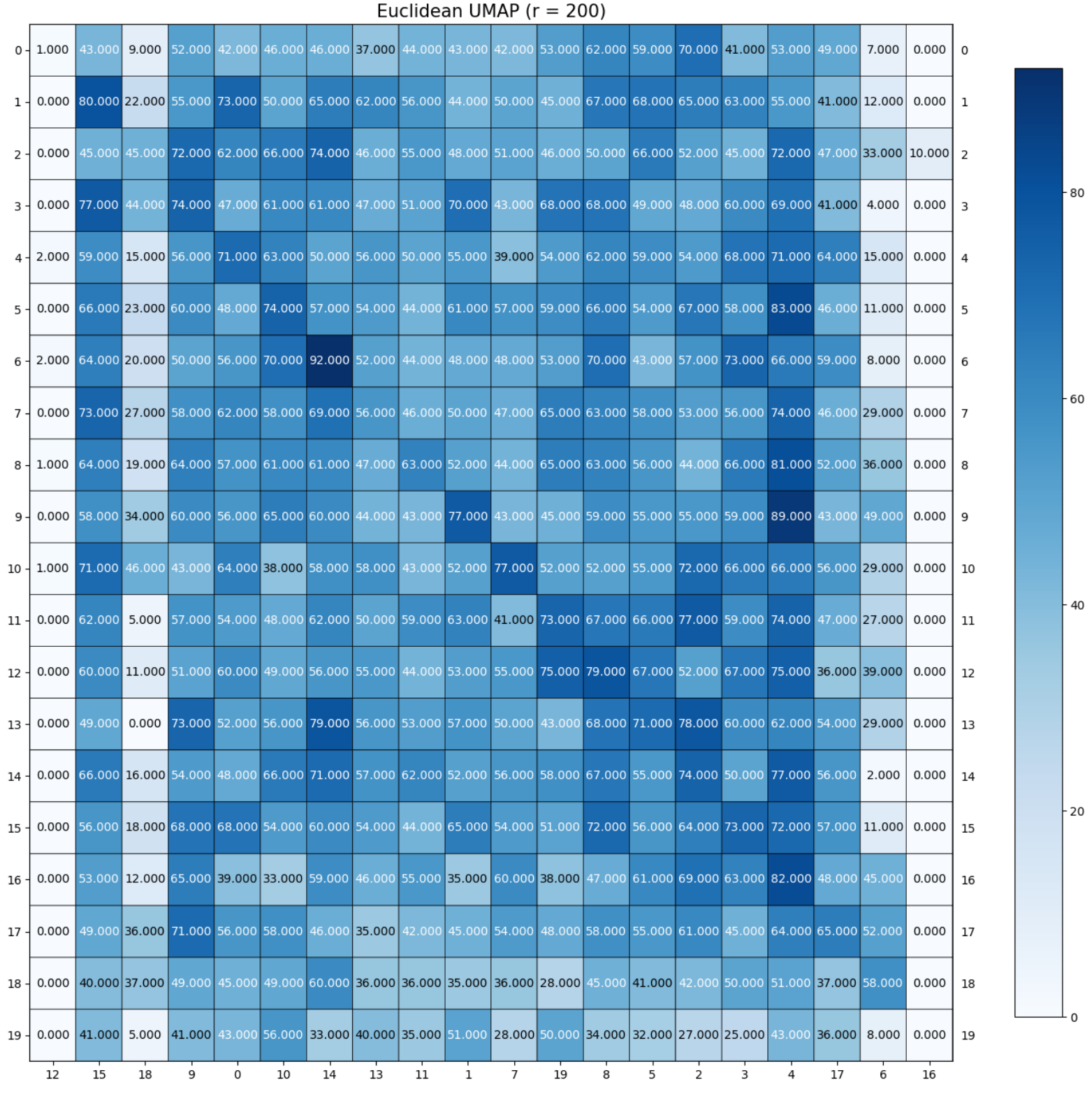

Cosine UMAP (r = 5)
0 540.000 12.000 0.000 1.000 1.000 0.000 1.000 1.000 5.000 1.000 2.000 1.000 0.000 9.000 7.000 180.000 9.000 19.000 10.000 0.000 0
1 3.000 205.000 115.000 43.000 4.000 554.000 11.000 7.000 4.000 2.000 2.000 2.000 0.000 6.000 7.000 2.000 5.000 1.000 0.000 0.000 1
2 2.000 49.000 540.000 87.000 5.000 242.000 21.000 4.000 2.000 1.000 4.000 2.000 10.000 2.000 8.000 2.000 4.000 0.000 0.000 0.000 2
3 5.000 54.000 110.000 686.000 2.000 28.000 72.000 3.000 1.000 3.000 2.000 2.000 0.000 3.000 8.000 0.000 2.000 0.000 1.000 0.000 3
4 5.000 49.000 65.000 646.000 3.000 61.000 78.000 16.000 5.000 6.000 2.000 3.000 0.000 2.000 12.000 2.000 5.000 1.000 2.000 0.000 4
5 2.000 84.000 70.000 16.000 1.000 778.000 11.000 4.000 2.000 2.000 1.000 1.000 0.000 0.000 7.000 4.000 4.000 0.000 0.000 1.000 5
6 8.000 126.000 39.000 212.000 7.000 30.000 432.000 49.000 7.000 21.000 7.000 5.000 0.000 8.000 10.000 5.000 5.000 1.000 3.000 0.000 6
7 8.000 57.000 3.000 8.000 2.000 9.000 14.000 765.000 59.000 2.000 5.000 1.000 0.000 16.000 16.000 1.000 15.000 3.000 6.000 0.000 7
8 8.000 44.000 3.000 11.000 1.000 3.000 31.000 67.000 785.000 4.000 6.000 2.000 0.000 3.000 3.000 5.000 18.000 1.000 1.000 0.000 8
9 8.000 52.000 1.000 1.000 3.000 3.000 5.000 21.000 15.000 785.000 57.000 2.000 0.000 20.000 5.000 4.000 10.000 0.000 2.000 0.000 9
10 1.000 28.000 0.000 2.000 1.000 2.000 10.000 7.000 1.000 20.000 914.000 1.000 0.000 1.000 2.000 1.000 4.000 4.000 0.000 0.000 10
11 6.000 36.000 11.000 6.000 0.000 16.000 8.000 7.000 2.000 1.000 1.000 827.000 0.000 4.000 1.000 3.000 60.000 2.000 0.000 0.000 11
12 12.000 186.000 53.000 137.000 0.000 42.000 361.000 78.000 11.000 6.000 4.000 6.000 0.000 29.000 46.000 5.000 4.000 2.000 1.000 1.000 12
13 33.000 84.000 9.000 7.000 2.000 16.000 17.000 2.000 23.000 2.000 0.000 4.000 0.000 690.000 57.000 12.000 24.000 4.000 3.000 1.000 13
14 7.000 41.000 2.000 4.000 1.000 21.000 7.000 16.000 6.000 5.000 4.000 2.000 0.000 16.000 324.000 8.000 22.000 1.000 0.000 0.000 14
15 36.000 34.000 2.000 2.000 0.000 4.000 1.000 1.000 2.000 1.000 1.000 2.000 0.000 13.000 1.000 827.000 13.000 15.000 40.000 2.000 15
16 15.000 25.000 2.000 1.000 0.000 2.000 8.000 11.000 8.000 2.000 2.000 10.000 0.000 12.000 8.000 8.000 785.000 4.000 7.000 0.000 16
17 11.000 21.000 1.000 1.000 0.000 3.000 5.000 3.000 9.000 2.000 0.000 1.000 0.000 5.000 1.000 12.000 46.000 611.000 1.000 207.000 17
18 20.000 18.000 5.000 0.000 0.000 1.000 5.000 6.000 10.000 3.000 3.000 6.000 0.000 70.000 19.000 7.000 386.000 25.000 189.000 2.000 18
19 184.000 19.000 3.000 1.000 2.000 3.000 1.000 0.000 2.000 12.000 1.000 1.000 0.000 23.000 2.000 261.000 82.000 7.000 22.000 2.000 19
3 4 14 13 9 6 1 2 11 17 5 7 19 8 12 16 10 0 15 18
800
600
400
200
0

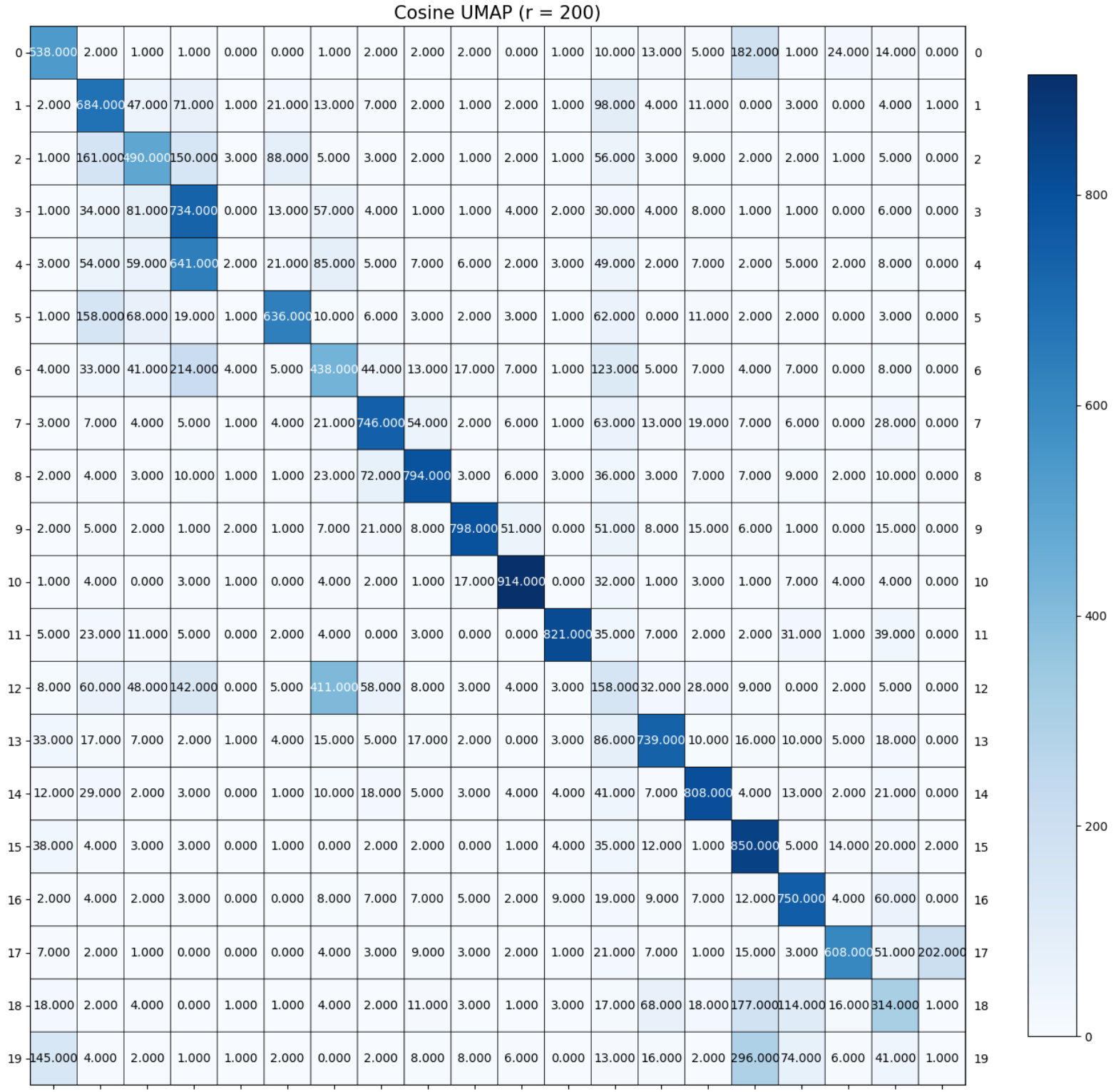
```
Rank :  20
```


Euclidean UMAP (r = 20)
17 13 12 2 19 14 7 11 3 0 6 16 4 8 5 9 10 15 1 18
0 1.000 80.000 0.000 55.000 48.000 47.000 58.000 58.000 33.000 37.000 50.000 33.000 64.000 39.000 39.000 45.000 56.000 39.000 17.000 0.000 0
1 0.000 99.000 0.000 63.000 59.000 43.000 75.000 65.000 59.000 62.000 45.000 11.000 72.000 57.000 49.000 61.000 57.000 65.000 31.000 0.000 1
2 0.000 67.000 10.000 62.000 51.000 46.000 68.000 72.000 66.000 68.000 46.000 42.000 56.000 50.000 54.000 48.000 60.000 69.000 50.000 0.000 2
3 0.000 63.000 0.000 75.000 54.000 72.000 82.000 58.000 60.000 84.000 44.000 3.000 69.000 52.000 39.000 55.000 58.000 67.000 47.000 0.000 3
4 2.000 64.000 0.000 60.000 55.000 58.000 67.000 69.000 61.000 61.000 44.000 23.000 68.000 59.000 58.000 69.000 70.000 60.000 15.000 0.000 4
5 0.000 83.000 0.000 66.000 52.000 70.000 59.000 56.000 67.000 75.000 49.000 12.000 74.000 55.000 51.000 70.000 83.000 59.000 7.000 0.000 5
6 2.000 73.000 0.000 58.000 56.000 62.000 90.000 57.000 61.000 61.000 45.000 4.000 74.000 47.000 58.000 77.000 68.000 54.000 21.000 7.000 6
7 0.000 83.000 0.000 57.000 51.000 62.000 61.000 76.000 62.000 75.000 44.000 24.000 70.000 53.000 46.000 63.000 66.000 58.000 39.000 0.000 7
8 1.000 54.000 0.000 67.000 49.000 55.000 84.000 70.000 63.000 66.000 32.000 39.000 63.000 48.000 58.000 74.000 71.000 61.000 41.000 0.000 8
9 0.000 58.000 0.000 56.000 57.000 64.000 55.000 59.000 66.000 86.000 45.000 24.000 76.000 50.000 59.000 55.000 74.000 54.000 56.000 0.000 9
10 1.000 76.000 0.000 56.000 56.000 51.000 51.000 60.000 56.000 79.000 53.000 47.000 72.000 39.000 57.000 70.000 52.000 52.000 71.000 0.000 10
11 0.000 97.000 0.000 59.000 48.000 75.000 62.000 80.000 41.000 51.000 35.000 64.000 57.000 62.000 56.000 60.000 73.000 50.000 21.000 0.000 11
12 1.000 62.000 0.000 51.000 57.000 57.000 62.000 80.000 60.000 47.000 49.000 39.000 91.000 52.000 52.000 71.000 66.000 51.000 36.000 0.000 12
13 0.000 94.000 0.000 66.000 42.000 47.000 83.000 62.000 46.000 61.000 57.000 39.000 59.000 72.000 61.000 58.000 62.000 65.000 16.000 0.000 13
14 0.000 93.000 0.000 66.000 48.000 61.000 71.000 66.000 68.000 61.000 48.000 19.000 67.000 46.000 70.000 58.000 73.000 56.000 16.000 0.000 14
15 0.000 69.000 0.000 72.000 64.000 66.000 59.000 55.000 59.000 59.000 55.000 8.000 70.000 53.000 60.000 84.000 74.000 68.000 22.000 0.000 15
16 0.000 77.000 0.000 55.000 53.000 48.000 65.000 56.000 32.000 43.000 50.000 48.000 49.000 53.000 68.000 57.000 87.000 52.000 17.000 0.000 16
17 0.000 71.000 0.000 68.000 47.000 65.000 54.000 50.000 63.000 59.000 51.000 34.000 56.000 42.000 57.000 49.000 56.000 63.000 55.000 0.000 17
18 0.000 52.000 0.000 34.000 40.000 44.000 65.000 39.000 48.000 48.000 34.000 32.000 45.000 51.000 42.000 39.000 48.000 46.000 68.000 0.000 18
19 0.000 47.000 0.000 45.000 41.000 60.000 41.000 39.000 52.000 35.000 26.000 15.000 43.000 23.000 27.000 26.000 50.000 45.000 13.000 0.000 19
80
60
40
20
0


Cosine UMAP (r = 20)
0 534.000 0.000 0.000 0.000 1.000 1.000 1.000 2.000 6.000 0.000 1.000 1.000 12.000 10.000 5.000 178.000 8.000 22.000 17.000 0.000 0
1 1.000 3.000 99.000 1.000 96.000 591.000 13.000 5.000 1.000 3.000 2.000 1.000 136.000 3.000 10.000 2.000 4.000 0.000 1.000 1.000 1
2 1.000 12.000 581.000 28.000 119.000 156.000 6.000 7.000 2.000 0.000 1.000 1.000 49.000 4.000 12.000 2.000 3.000 1.000 0.000 0.000 2
3 4.000 1.000 100.000 303.000 438.000 22.000 39.000 4.000 1.000 0.000 2.000 2.000 50.000 3.000 11.000 0.000 2.000 0.000 0.000 0.000 3
4 5.000 3.000 102.000 158.000 514.000 19.000 54.000 18.000 3.000 3.000 1.000 1.000 61.000 3.000 5.000 2.000 7.000 1.000 3.000 0.000 4
5 1.000 2.000 85.000 5.000 16.000 771.000 6.000 4.000 2.000 2.000 3.000 3.000 76.000 0.000 7.000 1.000 3.000 0.000 0.000 1.000 5
6 5.000 5.000 57.000 88.000 142.000 6.000 428.000 80.000 7.000 5.000 7.000 4.000 110.000 7.000 9.000 3.000 7.000 0.000 5.000 0.000 6
7 6.000 0.000 3.000 6.000 0.000 5.000 17.000 781.000 52.000 0.000 6.000 1.000 55.000 17.000 14.000 2.000 17.000 1.000 7.000 0.000 7
8 2.000 1.000 4.000 6.000 7.000 3.000 20.000 63.000 804.000 2.000 5.000 3.000 38.000 4.000 4.000 5.000 9.000 3.000 13.000 0.000 8
9 6.000 1.000 1.000 2.000 0.000 6.000 2.000 24.000 6.000 802.000 54.000 1.000 58.000 4.000 8.000 3.000 10.000 1.000 5.000 0.000 9
10 2.000 0.000 1.000 2.000 2.000 3.000 6.000 7.000 1.000 22.000 911.000 1.000 29.000 1.000 2.000 1.000 4.000 3.000 1.000 0.000 10
11 5.000 0.000 16.000 0.000 5.000 16.000 6.000 7.000 2.000 0.000 0.000 816.000 48.000 4.000 1.000 3.000 60.000 1.000 1.000 0.000 11
12 9.000 0.000 70.000 29.000 124.000 22.000 359.000 86.000 9.000 3.000 3.000 2.000 182.000 36.000 32.000 4.000 5.000 0.000 9.000 0.000 12
13 32.000 1.000 12.000 1.000 6.000 9.000 10.000 7.000 16.000 3.000 0.000 3.000 111.000 686.000 47.000 8.000 13.000 0.000 24.000 1.000 13
14 10.000 0.000 3.000 2.000 1.000 23.000 5.000 15.000 5.000 3.000 4.000 3.000 49.000 10.000 326.000 4.000 23.000 0.000 1.000 0.000 14
15 27.000 0.000 4.000 1.000 1.000 3.000 1.000 2.000 2.000 0.000 2.000 4.000 40.000 10.000 7.000 827.000 8.000 18.000 38.000 2.000 15
16 6.000 0.000 3.000 0.000 3.000 1.000 6.000 9.000 6.000 2.000 1.000 10.000 26.000 9.000 7.000 9.000 791.000 3.000 18.000 0.000 16
17 8.000 0.000 4.000 0.000 0.000 2.000 2.000 4.000 6.000 1.000 1.000 4.000 28.000 8.000 2.000 14.000 31.000 601.000 21.000 203.000 17
18 12.000 1.000 5.000 1.000 0.000 1.000 3.000 5.000 13.000 2.000 2.000 4.000 17.000 73.000 6.000 10.000 384.000 19.000 215.000 2.000 18
19 142.000 2.000 3.000 0.000 2.000 3.000 0.000 3.000 2.000 8.000 0.000 1.000 19.000 19.000 6.000 278.000 81.000 3.000 56.000 0.000 19
13 7 11 0 19 4 9 15 1 5 12 14 16 8 6 3 2 17 18 10
800
600
400
200
0

```
Rank :  200
```


Euclidean UMAP (r = 200)
0 1.000 43.000 9.000 52.000 42.000 46.000 46.000 37.000 44.000 43.000 42.000 53.000 62.000 59.000 70.000 41.000 53.000 49.000 7.000 0.000 0
1 0.000 80.000 22.000 55.000 73.000 50.000 65.000 62.000 56.000 44.000 50.000 45.000 67.000 68.000 65.000 63.000 55.000 41.000 12.000 0.000 1
2 0.000 45.000 45.000 72.000 62.000 66.000 74.000 46.000 55.000 48.000 51.000 46.000 50.000 66.000 52.000 45.000 72.000 47.000 33.000 10.000 2
3 0.000 77.000 44.000 74.000 47.000 61.000 61.000 47.000 51.000 70.000 43.000 68.000 68.000 49.000 48.000 60.000 69.000 41.000 4.000 0.000 3
4 2.000 59.000 15.000 56.000 71.000 63.000 50.000 56.000 50.000 55.000 39.000 54.000 62.000 59.000 54.000 68.000 71.000 64.000 15.000 0.000 4
5 0.000 66.000 23.000 60.000 48.000 74.000 57.000 54.000 44.000 61.000 57.000 59.000 66.000 54.000 67.000 58.000 83.000 46.000 11.000 0.000 5
6 2.000 64.000 20.000 50.000 56.000 70.000 92.000 52.000 44.000 48.000 48.000 53.000 70.000 43.000 57.000 73.000 66.000 59.000 8.000 0.000 6
7 0.000 73.000 27.000 58.000 62.000 58.000 69.000 56.000 46.000 50.000 47.000 65.000 63.000 58.000 53.000 56.000 74.000 46.000 29.000 0.000 7
8 1.000 64.000 19.000 64.000 57.000 61.000 61.000 47.000 63.000 52.000 44.000 65.000 63.000 56.000 44.000 66.000 81.000 52.000 36.000 0.000 8
9 0.000 58.000 34.000 60.000 56.000 65.000 60.000 44.000 43.000 77.000 43.000 45.000 59.000 55.000 55.000 59.000 89.000 43.000 49.000 0.000 9
10 1.000 71.000 46.000 43.000 64.000 38.000 58.000 58.000 43.000 52.000 77.000 52.000 52.000 55.000 72.000 66.000 66.000 56.000 29.000 0.000 10
11 0.000 62.000 5.000 57.000 54.000 48.000 62.000 50.000 59.000 63.000 41.000 73.000 67.000 66.000 77.000 59.000 74.000 47.000 27.000 0.000 11
12 0.000 60.000 11.000 51.000 60.000 49.000 56.000 55.000 44.000 53.000 55.000 75.000 79.000 67.000 52.000 67.000 75.000 36.000 39.000 0.000 12
13 0.000 49.000 0.000 73.000 52.000 56.000 79.000 56.000 53.000 57.000 50.000 43.000 68.000 71.000 78.000 60.000 62.000 54.000 29.000 0.000 13
14 0.000 66.000 16.000 54.000 48.000 66.000 71.000 57.000 62.000 52.000 56.000 58.000 67.000 55.000 74.000 50.000 77.000 56.000 2.000 0.000 14
15 0.000 56.000 18.000 68.000 68.000 54.000 60.000 54.000 44.000 65.000 54.000 51.000 72.000 56.000 64.000 73.000 72.000 57.000 11.000 0.000 15
16 0.000 53.000 12.000 65.000 39.000 33.000 59.000 46.000 55.000 35.000 60.000 38.000 47.000 61.000 69.000 63.000 82.000 48.000 45.000 0.000 16
17 0.000 49.000 36.000 71.000 56.000 58.000 46.000 35.000 42.000 45.000 54.000 48.000 58.000 55.000 61.000 45.000 64.000 65.000 52.000 0.000 17
18 0.000 40.000 37.000 49.000 45.000 49.000 60.000 36.000 36.000 35.000 36.000 28.000 45.000 41.000 42.000 50.000 51.000 37.000 58.000 0.000 18
19 0.000 41.000 5.000 41.000 43.000 56.000 33.000 40.000 35.000 51.000 28.000 50.000 34.000 32.000 27.000 25.000 43.000 36.000 8.000 0.000 19
12 15 18 9 0 10 14 13 11 1 7 19 8 5 2 3 4 17 6 16
80
60
40
20
0


Cosine UMAP (r = 200)
0 538.000 2.000 1.000 1.000 0.000 0.000 1.000 2.000 2.000 2.000 0.000 1.000 10.000 13.000 5.000 182.000 1.000 24.000 14.000 0.000 0
1 2.000 684.000 47.000 71.000 1.000 21.000 13.000 7.000 2.000 1.000 2.000 1.000 98.000 4.000 11.000 0.000 3.000 0.000 4.000 1.000 1
2 1.000 161.000 490.000 150.000 3.000 88.000 5.000 3.000 2.000 1.000 2.000 1.000 56.000 3.000 9.000 2.000 2.000 1.000 5.000 0.000 2
3 1.000 34.000 81.000 734.000 0.000 13.000 57.000 4.000 1.000 1.000 4.000 2.000 30.000 4.000 8.000 1.000 1.000 0.000 6.000 0.000 3
4 3.000 54.000 59.000 641.000 2.000 21.000 85.000 5.000 7.000 6.000 2.000 3.000 49.000 2.000 7.000 2.000 5.000 2.000 8.000 0.000 4
5 1.000 158.000 68.000 19.000 1.000 636.000 10.000 6.000 3.000 2.000 3.000 1.000 62.000 0.000 11.000 2.000 2.000 0.000 3.000 0.000 5
6 4.000 33.000 41.000 214.000 4.000 5.000 438.000 44.000 13.000 17.000 7.000 1.000 123.000 5.000 7.000 4.000 7.000 0.000 8.000 0.000 6
7 3.000 7.000 4.000 5.000 1.000 4.000 21.000 746.000 54.000 2.000 6.000 1.000 63.000 13.000 19.000 7.000 6.000 0.000 28.000 0.000 7
8 2.000 4.000 3.000 10.000 1.000 1.000 23.000 72.000 794.000 3.000 6.000 3.000 36.000 3.000 7.000 7.000 9.000 2.000 10.000 0.000 8
9 2.000 5.000 2.000 1.000 2.000 1.000 7.000 21.000 8.000 798.000 51.000 0.000 51.000 8.000 15.000 6.000 1.000 0.000 15.000 0.000 9
10 1.000 4.000 0.000 3.000 1.000 0.000 4.000 2.000 1.000 17.000 914.000 0.000 32.000 1.000 3.000 1.000 7.000 4.000 4.000 0.000 10
11 5.000 23.000 11.000 5.000 0.000 2.000 4.000 0.000 3.000 0.000 0.000 821.000 35.000 7.000 2.000 2.000 31.000 1.000 39.000 0.000 11
12 8.000 60.000 48.000 142.000 0.000 5.000 411.000 58.000 8.000 3.000 4.000 3.000 158.000 32.000 28.000 9.000 0.000 2.000 5.000 0.000 12
13 33.000 17.000 7.000 2.000 1.000 4.000 15.000 5.000 17.000 2.000 0.000 3.000 86.000 739.000 10.000 16.000 10.000 5.000 18.000 0.000 13
14 12.000 29.000 2.000 3.000 0.000 1.000 10.000 18.000 5.000 3.000 4.000 4.000 41.000 7.000 808.000 4.000 13.000 2.000 21.000 0.000 14
15 38.000 4.000 3.000 3.000 0.000 1.000 0.000 2.000 2.000 0.000 1.000 4.000 35.000 12.000 1.000 850.000 5.000 14.000 20.000 2.000 15
16 2.000 4.000 2.000 3.000 0.000 0.000 8.000 7.000 7.000 5.000 2.000 9.000 19.000 9.000 7.000 12.000 750.000 4.000 60.000 0.000 16
17 7.000 2.000 1.000 0.000 0.000 0.000 4.000 3.000 9.000 3.000 2.000 1.000 21.000 7.000 1.000 15.000 3.000 608.000 51.000 202.000 17
18 18.000 2.000 4.000 0.000 1.000 1.000 4.000 2.000 11.000 3.000 1.000 3.000 17.000 68.000 18.000 177.000 114.000 16.000 314.000 1.000 18
19 145.000 4.000 2.000 1.000 1.000 2.000 0.000 2.000 8.000 8.000 6.000 0.000 13.000 16.000 2.000 296.000 74.000 6.000 41.000 1.000 19
2 0 16 7 6 12 3 10 4 9 15 19 14 1 11 18 8 13 5 17
800
600
400
200
0

ANSWER-11

Contingency matrices are given as above. Metric scores for different ranks and U-Map are given below

Metric	Rank=5	Rank=20	Rank=200
UMAP (Euclidean)			
Homogeneity Score	0.007	0.008	0.007
Completeness Score	0.007	0.008	0.007
V-measure Score	0.007	0.008	0.007
Adjusted Rand Index Score	0.001	0.001	0.001
Adjusted Mutual Information Score	0.004	0.004	0.004
---	---	---	---
UMAP (Cosine)			
Homogeneity Score	0.568	0.566	0.566
Completeness Score	0.595	0.592	0.592
V-measure Score	0.581	0.579	0.578
Adjusted Rand Index Score	0.443	0.447	0.440
Adjusted Mutual Information Score	0.580	0.577	0.577

ANSWER-12

We observe the best rank in the averaging of all metrics for rank = 5 and metric = 'cosine'. We note that the contingency matrices are very bad for 'euclidean'. Contrastive to euclidean, cosine contingency matrices are much better with more diagonal elemnts. This is owing to the curse of dimensionality, where points become more uniformly distributed in high-dimensional spaces, and the Euclidean distance may lose its discriminatory power. Cosine metric tends to perform better than Euclidean distance in high-dimensional spaces because it is less affected by the curse of dimensionality and focuses on the angular relationships between vectors rather than their magnitudes.

ANSWER-13

All results here are mentioned below. We clearly see that choosing UMAP ('Cosine') gives best results when we consider the average of all 5 metrics as the decider.

	Homogeneity Score	Completeness Score	V-Measure Score	Adjusted Rand Index	Adjusted Mutual Information Score
SVD Truncated Scores	0.323	0.352	0.337	0.127	0.335
NMF Truncated Scores	0.301	0.339	0.319	0.103	0.317

Metric	Rank=5	Rank=20	Rank=200
UMAP (Euclidean)			
Homogeneity Score	0.007	0.008	0.007
Completeness Score	0.007	0.008	0.007
V-measure Score	0.007	0.008	0.007
Adjusted Rand Index Score	0.001	0.001	0.001
Adjusted Mutual Information Score	0.004	0.004	0.004
----------	----------	----------	----------
UMAP (Cosine)			
Homogeneity Score	0.568	0.566	0.566
Completeness Score	0.595	0.592	0.592
V-measure Score	0.581	0.579	0.578
Adjusted Rand Index Score	0.443	0.447	0.440

Agglomerative clustering

In [ ]:
```
# r =5 umap cosine
best_r = 5
best_umap_cos = umap.UMAP(n_components=best_r, metric='cosine').fit_transform(tfidf_20)
```

In [ ]:
```
agg_c_ward = AgglomerativeClustering(n_clusters=20, linkage='ward').fit(best_umap_cos)
agg_c_single = AgglomerativeClustering(n_clusters=20, linkage='single').fit(best_umap_cos)
print("Homogeneity (Agglomerative Clustering-Ward): %0.3f" % homogeneity_score(dataset.targe
t, agg_c_ward.labels_))
print("Completeness (Agglomerative Clustering- Ward): %0.3f" % completeness_score(dataset.ta
rget, agg_c_ward.labels_))
print("V-measure (Agglomerative Clustering- Ward): %0.3f" % v_measure_score(dataset.target,
agg_c_ward.labels_))
print("Adjusted Rand-Index (Agglomerative Clustering- Ward): %.3f"% adjusted_rand_score(data
set.target, agg_c_ward.labels_))
print("Adjusted Mutual Information Score (Agglomerative Clustering- Ward): %.3f"% adjusted_m
utual_info_score(dataset.target, agg_c_ward.labels_))
print("Homogeneity (Agglomerative Clustering-Single): %0.3f" % homogeneity_score(dataset.tar
get, agg_c_single.labels_))
print("Completeness (Agglomerative Clustering-Single): %0.3f" % completeness_score(dataset.t
arget, agg_c_single.labels_))
print("V-measure (Agglomerative Clustering-Single): %0.3f" % v_measure_score(dataset.target,
agg_c_single.labels_))
print("Adjusted Rand-Index (Agglomerative Clustering-Single): %.3f"% adjusted_rand_score(dat
aset.target, agg_c_single.labels_))
print("Adjusted Mutual Information Score (Agglomerative Clustering-Single): %.3f"% adjusted_
mutual_info_score(dataset.target, agg_c_single.labels_))
```

```
Homogeneity (Agglomerative Clustering-Ward): 0.547
Completeness (Agglomerative Clustering- Ward): 0.582
V-measure (Agglomerative Clustering- Ward): 0.564
Adjusted Rand-Index (Agglomerative Clustering- Ward): 0.421
Adjusted Mutual Information Score (Agglomerative Clustering- Ward): 0.563

Homogeneity (Agglomerative Clustering-Single): 0.016
Completeness (Agglomerative Clustering-Single): 0.365
V-measure (Agglomerative Clustering-Single): 0.030
Adjusted Rand-Index (Agglomerative Clustering-Single): 0.000
Adjusted Mutual Information Score (Agglomerative Clustering-Single): 0.025
```

Answer 14

Metric	Agglomerative Clustering (Ward)	Agglomerative Clustering (Single)
Homogeneity Score	0.547	0.016
Completeness Score	0.582	0.365
V-Measure Score	0.564	0.030
Adjusted Rand Index Score	0.421	0.000

Adjusted Mutual Information Score	0.563	0.025

HDBSCAN

In [ ]:
```
cluster_sizes_rec = []
min_samples_rec = []
hdb_hs = []
hdb_cs = []
hdb_vs = []
hdb_ars = []
hdb_amis = []


cluster_sizes = [20,100,200]
min_samples = [5,25,50,100,200,500,1000,3000]

for i in range(len(cluster_sizes)):
    for j in range(len(min_samples)):
        print('Hyperparameter search for cluster_size = {} & min_sample = {} '.format(cluste
r_sizes[i],min_samples[j]))
        hdbs = hdbscan.HDBSCAN(min_cluster_size=cluster_sizes[i], min_samples=min_samples
[j]).fit_predict(best_umap_cos)
        hdb_hs.append(homogeneity_score(dataset.target, hdbs))
        hdb_cs.append(completeness_score(dataset.target, hdbs))
        hdb_vs.append(v_measure_score(dataset.target, hdbs))
        hdb_ars.append(adjusted_rand_score(dataset.target, hdbs))
        hdb_amis.append(adjusted_mutual_info_score(dataset.target, hdbs))
        cluster_sizes_rec.append(cluster_sizes[i])
        min_samples_rec.append(min_samples[j])
print('Done testing')
```

```
Hyperparameter search for cluster_size = 20 & min_sample = 5
Hyperparameter search for cluster_size = 20 & min_sample = 25
Hyperparameter search for cluster_size = 20 & min_sample = 50
Hyperparameter search for cluster_size = 20 & min_sample = 100
Hyperparameter search for cluster_size = 20 & min_sample = 200
Hyperparameter search for cluster_size = 20 & min_sample = 500
Hyperparameter search for cluster_size = 20 & min_sample = 1000
Hyperparameter search for cluster_size = 20 & min_sample = 3000
Hyperparameter search for cluster_size = 100 & min_sample = 5
Hyperparameter search for cluster_size = 100 & min_sample = 25
Hyperparameter search for cluster_size = 100 & min_sample = 50
Hyperparameter search for cluster_size = 100 & min_sample = 100
Hyperparameter search for cluster_size = 100 & min_sample = 200
Hyperparameter search for cluster_size = 100 & min_sample = 500
Hyperparameter search for cluster_size = 100 & min_sample = 1000
Hyperparameter search for cluster_size = 100 & min_sample = 3000
Hyperparameter search for cluster_size = 200 & min_sample = 5
Hyperparameter search for cluster_size = 200 & min_sample = 25
Hyperparameter search for cluster_size = 200 & min_sample = 50
Hyperparameter search for cluster_size = 200 & min_sample = 100
Hyperparameter search for cluster_size = 200 & min_sample = 200
Hyperparameter search for cluster_size = 200 & min_sample = 500
Hyperparameter search for cluster_size = 200 & min_sample = 1000
Hyperparameter search for cluster_size = 200 & min_sample = 3000
Done testing
```

In [ ]:

```
average_metrics = [y/5 for y in [sum(x) for x in zip(hdb_hs, hdb_cs, hdb_vs, hdb_ars, hdb_amis)]]

best_cluster_size_hdb = cluster_sizes_rec[average_metrics.index(max(average_metrics))]
best_min_samples_hdb = min_samples_rec[average_metrics.index(max(average_metrics))]

print('Best Cluster size = {}, Best Minimum number of samples (HDBSCAN) = {}'.format(best_cluster_size_hdb, best_min_samples_hdb))
print('Average value of 5 metrics:', max(average_metrics))

print('\n------------ Best metrics ------------\n')
print('Homogeneity :', round(hdb_hs[average_metrics.index(max(average_metrics))],3))
print('Completeness :', round(hdb_cs[average_metrics.index(max(average_metrics))],3))
print('V-measure :', round(hdb_vs[average_metrics.index(max(average_metrics))],3))
print('Adjusted Rand-Index :', round(hdb_ars[average_metrics.index(max(average_metrics))],3))
print('Adjusted Mutual Information Score :', round(hdb_amis[average_metrics.index(max(average_metrics))],3))
```

```
Best Cluster size = 200, Best Minimum number of samples (HDBSCAN) = 25
Average value of 5 metrics: 0.4579520632745163

------------ Best metrics ------------

Homogeneity : 0.428
Completeness : 0.624
V-measure : 0.508
Adjusted Rand-Index : 0.223
Adjusted Mutual Information Score : 0.507
```

ANSWER-15

The best hyperparameters for HDBSCAN clustering were found to be a cluster size = 200 and min_samples = 25. The average value of five clustering metrics = 0.4579.

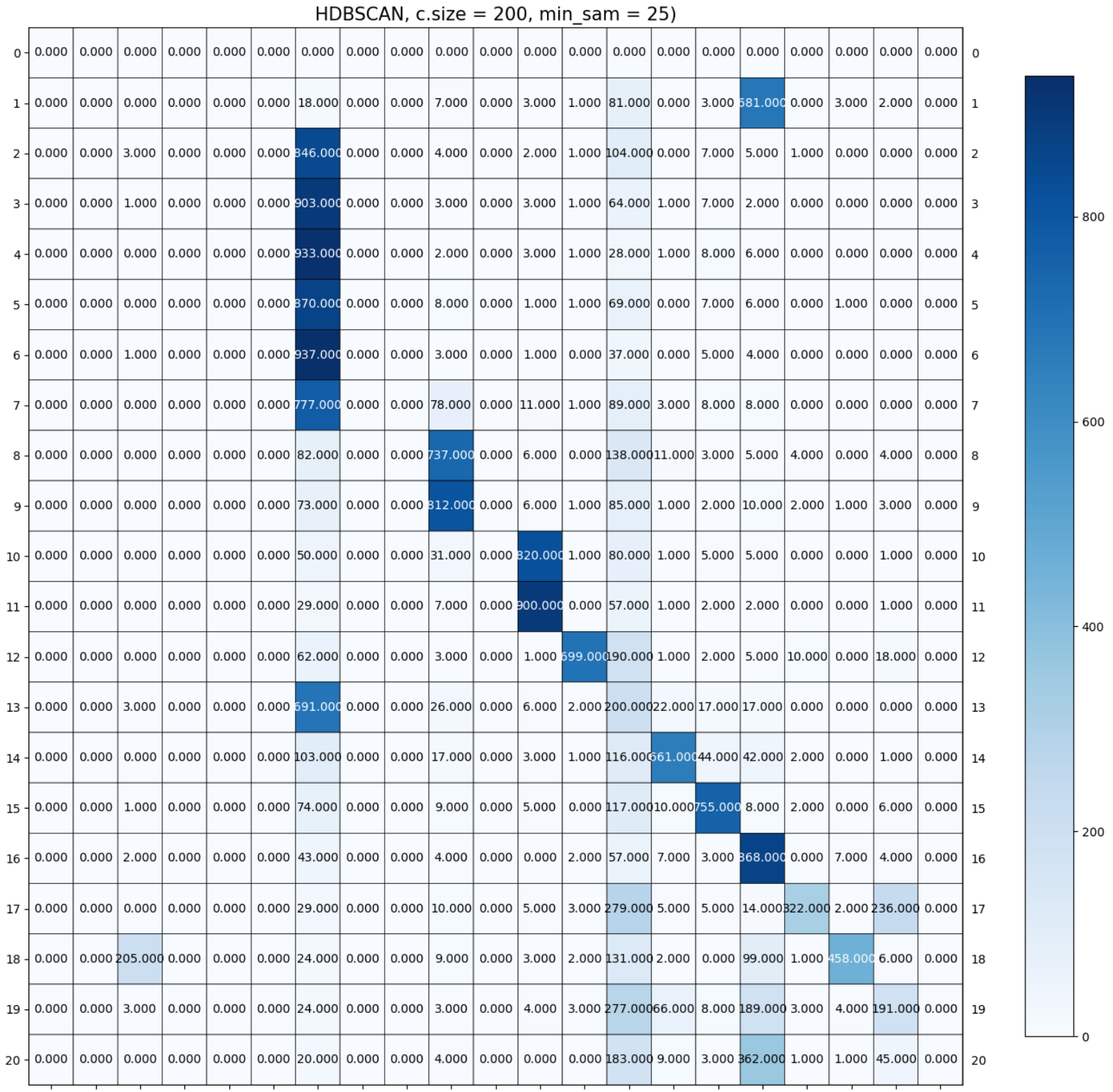
Best Metrics:

Metric	Value
Homogeneity	0.428
Completeness	0.624
V-measure	0.508
Adjusted Rand-Index	0.223
Adjusted Mutual Information	0.507

ANSWER-16

The contingence matrix for best hyperparameters (in our case cluster size = 200 and min_samples = 25) is plotted below

In [ ]:
```
hdbs = hdbscan.HDBSCAN(min_cluster_size=best_cluster_size_hdb,min_samples=best_min_samples_h
db).fit_predict(best_umap_cos)
cm = confusion_matrix(dataset.target, hdbs)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, title = 'HDBSCA
N, c.size = 200, min_sam = 25)', size=(13,13))
```


HDBSCAN, c.size = 200, min_sam = 25)
0 0.000 0
1 0.000 0.000 0.000 0.000 0.000 0.000 18.000 0.000 0.000 7.000 0.000 3.000 1.000 81.000 0.000 3.000 681.000 0.000 3.000 2.000 0.000 1
2 0.000 0.000 3.000 0.000 0.000 0.000 346.000 0.000 0.000 4.000 0.000 2.000 1.000 104.000 0.000 7.000 5.000 1.000 0.000 0.000 0.000 2
3 0.000 0.000 1.000 0.000 0.000 0.000 903.000 0.000 0.000 3.000 0.000 3.000 1.000 64.000 1.000 7.000 2.000 0.000 0.000 0.000 0.000 3
4 0.000 0.000 0.000 0.000 0.000 0.000 933.000 0.000 0.000 2.000 0.000 3.000 1.000 28.000 1.000 8.000 6.000 0.000 0.000 0.000 0.000 4
5 0.000 0.000 0.000 0.000 0.000 0.000 870.000 0.000 0.000 8.000 0.000 1.000 1.000 69.000 0.000 7.000 6.000 0.000 1.000 0.000 0.000 5
6 0.000 0.000 1.000 0.000 0.000 0.000 937.000 0.000 0.000 3.000 0.000 1.000 0.000 37.000 0.000 5.000 4.000 0.000 0.000 0.000 0.000 6
7 0.000 0.000 0.000 0.000 0.000 0.000 777.000 0.000 0.000 78.000 0.000 11.000 1.000 89.000 3.000 8.000 8.000 0.000 0.000 0.000 0.000 7
8 0.000 0.000 0.000 0.000 0.000 0.000 82.000 0.000 0.000 737.000 0.000 6.000 0.000 138.000 11.000 3.000 5.000 4.000 0.000 4.000 0.000 8
9 0.000 0.000 0.000 0.000 0.000 0.000 73.000 0.000 0.000 812.000 0.000 6.000 1.000 85.000 1.000 2.000 10.000 2.000 1.000 3.000 0.000 9
10 0.000 0.000 0.000 0.000 0.000 0.000 50.000 0.000 0.000 31.000 0.000 820.000 1.000 80.000 1.000 5.000 5.000 0.000 0.000 1.000 0.000 10
11 0.000 0.000 0.000 0.000 0.000 0.000 29.000 0.000 0.000 7.000 0.000 900.000 0.000 57.000 1.000 2.000 2.000 0.000 0.000 1.000 0.000 11
12 0.000 0.000 0.000 0.000 0.000 0.000 62.000 0.000 0.000 3.000 0.000 1.000 699.000 190.000 1.000 2.000 5.000 10.000 0.000 18.000 0.000 12
13 0.000 0.000 3.000 0.000 0.000 0.000 691.000 0.000 0.000 26.000 0.000 6.000 2.000 200.000 22.000 17.000 17.000 0.000 0.000 0.000 0.000 13
14 0.000 0.000 0.000 0.000 0.000 0.000 103.000 0.000 0.000 17.000 0.000 3.000 1.000 116.000 661.000 44.000 42.000 2.000 0.000 1.000 0.000 14
15 0.000 0.000 1.000 0.000 0.000 0.000 74.000 0.000 0.000 9.000 0.000 5.000 0.000 117.000 10.000 755.000 8.000 2.000 0.000 6.000 0.000 15
16 0.000 0.000 2.000 0.000 0.000 0.000 43.000 0.000 0.000 4.000 0.000 0.000 2.000 57.000 7.000 3.000 868.000 0.000 7.000 4.000 0.000 16
17 0.000 0.000 0.000 0.000 0.000 0.000 29.000 0.000 0.000 10.000 0.000 5.000 3.000 279.000 5.000 5.000 14.000 322.000 2.000 236.000 0.000 17
18 0.000 0.000 205.000 0.000 0.000 0.000 24.000 0.000 0.000 9.000 0.000 3.000 2.000 131.000 2.000 0.000 99.000 1.000 458.000 6.000 0.000 18
19 0.000 0.000 3.000 0.000 0.000 0.000 24.000 0.000 0.000 3.000 0.000 4.000 3.000 277.000 66.000 8.000 189.000 3.000 4.000 191.000 0.000 19
20 0.000 0.000 0.000 0.000 0.000 0.000 20.000 0.000 0.000 4.000 0.000 0.000 0.000 183.000 9.000 3.000 362.000 1.000 1.000 45.000 0.000 20
14 18 1 12 15 16 6 13 19 4 17 2 9 0 3 5 8 11 7 10 20
800
600
400
200
0

ANSWER-16

We note from the contingency matrix above that HDBSCAN has produced a total of around 10 major clusters. Many ground truth clusters that were initially missing have been merged into other clusters because the clustering is organized as a tree structure rather than a flat one.

The label "-1" in the clustering results denotes outliers or samples that haven't been assigned to any specific cluster by the algorithms. The merging of clusters is likely attributed to excessive smoothing, influenced by hyperparameters' sensitivity, while the minimum cluster size remains unchanged. This has led to the loss of low-density clusters. Additionally, both algorithms struggle to identify clusters that exhibit significant variation or are sparse in high-dimensional spaces, a scenario common in textual data.

Best dim-reduction and clustering algorithm

In [5]:
```
def custom_scorer(y_true, labels):
    """
    Custom scoring function that combines multiple clustering metrics.

    Parameters:
    - y_true: True labels (ground truth).
    - labels: Predicted labels from the clustering algorithm.

    Returns:
    - Average of homogeneity score, completeness score, V-measure score, adjusted Rand index score,
      and adjusted mutual information score.
    """
    homogeneity = homogeneity_score(y_true, labels)
    completeness = completeness_score(y_true, labels)
    v_measure = v_measure_score(y_true, labels)
    adjusted_rand = adjusted_rand_score(y_true, labels)
    adjusted_mutual_info = adjusted_mutual_info_score(y_true, labels)

    # Calculate the average of the scores
    average_score = (homogeneity + completeness + v_measure + adjusted_rand + adjusted_mutual_info) / 5.0

    return average_score
```

In [ ]:

```
from sklearn.model_selection import GridSearchCV,ShuffleSplit
from sklearn.pipeline import Pipeline

dataset = fetch_20newsgroups(subset = 'all',shuffle = True, random_state = 42,remove=('heade
rs','footers'))
dvec_20 = CountVectorizer(stop_words='english',min_df=3)
tfidf_20 = TfidfTransformer()
data_dvec_20 = dvec_20.fit_transform(dataset.data)
tfidf_20 = tfidf_20.fit_transform(data_dvec_20)
y_true = dataset.target

# Define hyperparameter combinations for dimensionality reduction
dimensionality_reduction_methods = {'UMAP': umap.UMAP,'SVD': TruncatedSVD, 'NMF': NMF}
dimensionality_reduction_params = {'UMAP': [5, 20, 200],'SVD': [5, 20, 200], 'NMF': [5, 20,
200]}

# Define hyperparameter combinations for clustering
clustering_methods = {'Agglomerative': AgglomerativeClustering,'KMeans': KMeans, 'HDBSCAN':
hdbscan.HDBSCAN}
clustering_params = {'Agglomerative': [20],'KMeans': [10, 20, 50], 'HDBSCAN': [100, 200]}

best_score = float('-inf')
best_params = {}
# Iterate through hyperparameter combinations
for dim_reduction_method, dim_reduction_param_list in dimensionality_reduction_params.items
():
    for clustering_method, clustering_param_list in clustering_params.items():
        i=1
        print("---> Running {} with {}  ..... ".format(dim_reduction_me
thod,clustering_method))
        for dim_reduction_param in dim_reduction_param_list:
            for clustering_param in clustering_param_list:
                print("Run # : ", i); i+=1

                # Apply dimensionality reduction

                if dim_reduction_method == 'UMAP':
                  dim_reduction_model = umap.UMAP(n_components=dim_reduction_param, metric
='cosine')
                else:
                  dim_reduction_model = dimensionality_reduction_methods[dim_reduction_metho
d](n_components=dim_reduction_param)

                features = dim_reduction_model.fit_transform(tfidf_20)
```

```
                if clustering_method == 'HDBSCAN':
                    clustering_model = hdbscan.HDBSCAN(min_cluster_size=clustering_param, mi
n_samples=30)
                elif clustering_method == 'Agglomerative':
                    clustering_model = AgglomerativeClustering(n_clusters=clustering_param,
linkage='ward')
                else:
                    clustering_model = clustering_methods[clustering_method](n_clusters=clus
tering_param)

                labels = clustering_model.fit_predict(features)

                # Calculate custom score
                custom_score = custom_scorer(y_true, labels)

                # Update best parameters if the current combination is better
                if custom_score > best_score:
                    best_score = custom_score
                    best_params = {
                        'dim_reduction_method': dim_reduction_method,
                        'dim_reduction_param': dim_reduction_param,
                        'clustering_method': clustering_method,
                        'clustering_param': clustering_param
                    }

# Print the best parameters and corresponding score
print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
---> Running UMAP with Agglomerative  ........................
Run # :  1
Run # :  2
Run # :  3
---> Running UMAP with KMeans  ........................
Run # :  1
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  2
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  3
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  4
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  5
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  6
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  7
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  8
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
Run # :  9
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
---> Running UMAP with HDBSCAN  ........................
Run # :  1
Run # :  2
Run # :  3
Run # :  4
Run # :  5
Run # :  6
---> Running SVD with Agglomerative  ......................
Run # :  1
Run # :  2
Run # :  3
---> Running SVD with KMeans  .....................
Run # :  1
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  2
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  3
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  4
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  5
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  6
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  7
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  8
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  9
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
---> Running SVD with HDBSCAN  ........................
Run # :  1
Run # :  2
Run # :  3
Run # :  4
Run # :  5
Run # :  6
---> Running NMF with Agglomerative  ........................
Run # :  1
Run # :  2
Run # :  3
---> Running NMF with KMeans  ........................
Run # :  1
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  2
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  3
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  4
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  5
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  6
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  7
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWarnin
g: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  8
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Run # :  9
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
---> Running NMF with HDBSCAN  ........................
Run # :  1
Run # :  2
Run # :  3
Run # :  4
Run # :  5
Run # :  6
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWarnin
g: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```

```
Best Parameters: {'dim_reduction_method': 'UMAP', 'dim_reduction_param': 20, 'clustering_meth
od': 'KMeans', 'clustering_param': 20}
Best Score: 0.5565803679648941
```

In [ ]:
```
best_params

print("------------------- BEST DIM_RED + CLUSTERING ------------------- \n")

print("Best dimension reduction : " , best_params['dim_reduction_method'], "   |  Rank = ",
best_params['dim_reduction_param'])
print("Best clustering          : ", best_params['clustering_method'], " |  Number of cluste
rs = ", best_params['clustering_param'])
print("-----------------------------------------------------------------")
```

```
------------------- BEST DIM_RED + CLUSTERING -------------------

Best dimension reduction :  UMAP    |  Rank =  20
Best clustering          :  KMeans  |  Number of clusters =  20
-----------------------------------------------------------------
```

ANSWER-17

For this question, I used a custom scorer that takes the average of the 5 scores and hence finds the best combination. The best combination found is as in the above table of the code cell output.

Best Dimension Reduction:

- Technique: UMAP
- Rank: 20

Best Clustering:

- Method: KMeans
- Number of Clusters: 20

Best Score

Custom score = 0.557

We note that UMap and KMeans go hand-in-hand. UMAP being a Non-linear Dimension Reduction technique is effective for non-linear dimension reduction, preserving both local and global structure in the data, and can capture complex relationships in high-dimensional data, making it suitable for our dataset. UMAP tends to preserve the inherent cluster structures in the data, making it useful for applications where identifying natural groupings or clusters is important, and thereon KMeans finds the best cluster centroids.

ANSWER-18

I tried k-means++ as initializer and tried standardizing data, but it gave results that are either equal or subpar to the above best case custom_score= 0.557.

Student details

NAME : Vignesh Nagarajan

UID: 606185377

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
!pip uninstall umap
!pip install umap-learn
!pip install hdbscan
```

```
WARNING: Skipping umap as it is not installed.
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.9/90.9 kB 3.2 MB/
s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from u
map-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from
umap-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (1.2.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-packages (from
umap-learn) (0.58.1)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.8/55.8 kB 7.8 MB/
s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from umap-lea
rn) (4.66.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-p
ackages (from numba>=0.51.2->umap-learn) (0.41.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from
pynndescent>=0.5->umap-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn>=0.22->umap-learn) (3.2.0)
Building wheels for collected packages: umap-learn
  Building wheel for umap-learn (setup.py) ... done
  Created wheel for umap-learn: filename=umap_learn-0.5.5-py3-none-any.whl size=86832 sha256=
a585d2adb41a2cb351e9de73eaa0b6ec9c0e23fc220ed41ca87667bb2761510f
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a3b431db59b806a10da73661
2ebbc66c1bcc5
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
Collecting hdbscan
  Downloading hdbscan-0.8.33.tar.gz (5.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.2/5.2 MB 22.6 MB/s
eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting cython<3,>=0.27 (from hdbscan)
  Using cached Cython-0.29.37-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinu
x_2_24_x86_64.whl (1.9 MB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from h
```

```
dbscan) (1.23.5)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from hd
bscan) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dist-packages
(from hdbscan) (1.2.2)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.10/dist-packages (from h
dbscan) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-package
s (from scikit-learn>=0.20->hdbscan) (3.2.0)
Building wheels for collected packages: hdbscan
  Building wheel for hdbscan (pyproject.toml) ... done
  Created wheel for hdbscan: filename=hdbscan-0.8.33-cp310-cp310-linux_x86_64.whl size=303929
5 sha256=40624b67c609c0e5a985d11a46a6f4de958da4f9439e947f814dc1d092fd4633
  Stored in directory: /root/.cache/pip/wheels/75/0b/3b/dc4f60b7cc455efaefb62883a7483e76f09d0
6ca81cf87d610
Successfully built hdbscan
Installing collected packages: cython, hdbscan
  Attempting uninstall: cython
    Found existing installation: Cython 3.0.8
    Uninstalling Cython-3.0.8:
      Successfully uninstalled Cython-3.0.8
Successfully installed cython-0.29.37 hdbscan-0.8.33
```

In [ ]:
```
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 sklearn.model_selection import train_test_split

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_scor
e
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin


from sklearn.metrics.cluster import homogeneity_score,\
                                    completeness_score,\
                                    v_measure_score,\
                                    adjusted_rand_score,\
                                    adjusted_mutual_info_score
from sklearn.cluster import KMeans,  AgglomerativeClustering, DBSCAN
from tqdm import tqdm
import umap.umap_ as umap
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import contingency_matrix
import hdbscan
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
```

In [ ]:
```
%cd '/content/drive/MyDrive/219/Project2'
```

```
/content/drive/MyDrive/219/Project2
```

ANSWER-19

When a VGG network is trained on a set of images with different types of objects, it learns to recognize generic visual features like edges, textures, etc. If we want to use this pre-trained network for a new set of images (our flowers dataset), it turns out

that the features it learned can still be pretty useful because of the power of transfer learning. It's like the network learned to see the basics of visual information that can be handy in different scenarios. So, even if the initial training was on something completely different, those learned features can give a head start when working with your own set of pictures, making it easier to train a model that can recognize specific things in our flower images.

Flowers Dataset and VGG Features

In [ ]:

```
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], std=[0.229, 0.224, 0.225])]))
    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)
```

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.cache/torch/hub/v
0.10.0.zip
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The pa
rameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'w
eights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Argume
nts other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be rem
oved in the future. The current behavior is equivalent to passing `weights=VGG16_Weights.IMAG
ENET1K_V1`. You can also use `weights=VGG16_Weights.DEFAULT` to get the most up-to-date weigh
ts.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/h
ub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:07<00:00, 73.6MB/s]
100%|██████████| 58/58 [00:50<00:00,  1.14it/s]
```

In [ ]:

```
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], std=[0.229, 0.224, 0.225])]))
    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)
```

ANSWER-20

The code performs feature extraction using the VGG16 model on the dataset of flower images. The features are then saved to a NumPy compressed file for future use.

Loading Features and Labels

The code checks if a file named 'flowers_features_and_labels.npz' exists. If it does, it loads the features (`f_all`) and corresponding labels (`y_all`) from the file. If not, it proceeds to download the flower dataset, extract the images, and perform feature extraction.

Feature Extraction Process

1. **Model Initialization:** The `FeatureExtractor` class is defined, which is a PyTorch module utilizing the pre-trained VGG16 model. It extracts features from the convolutional layers, performs average pooling, and flattens the output.
2. **Model Initialization:** The model is initialized,
3. **Dataset Loading:** The flower dataset is loaded and transformed to meet the input requirements of VGG16.
4. **Feature Extraction:** A loop iterates through the dataset (`dataloader`), extracts features using the `FeatureExtractor` model, and appends them to the `f_all` array. Corresponding labels are also stored in the `y_all` array, and we save these results.

In [ ]:
```
print(f_all.shape, y_all.shape)
num_features = f_all.shape[1]
```

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

ANSWER-21

The TensorFlow Flowers dataset contains images of varying sizes. We note that `num_features` is 4096.

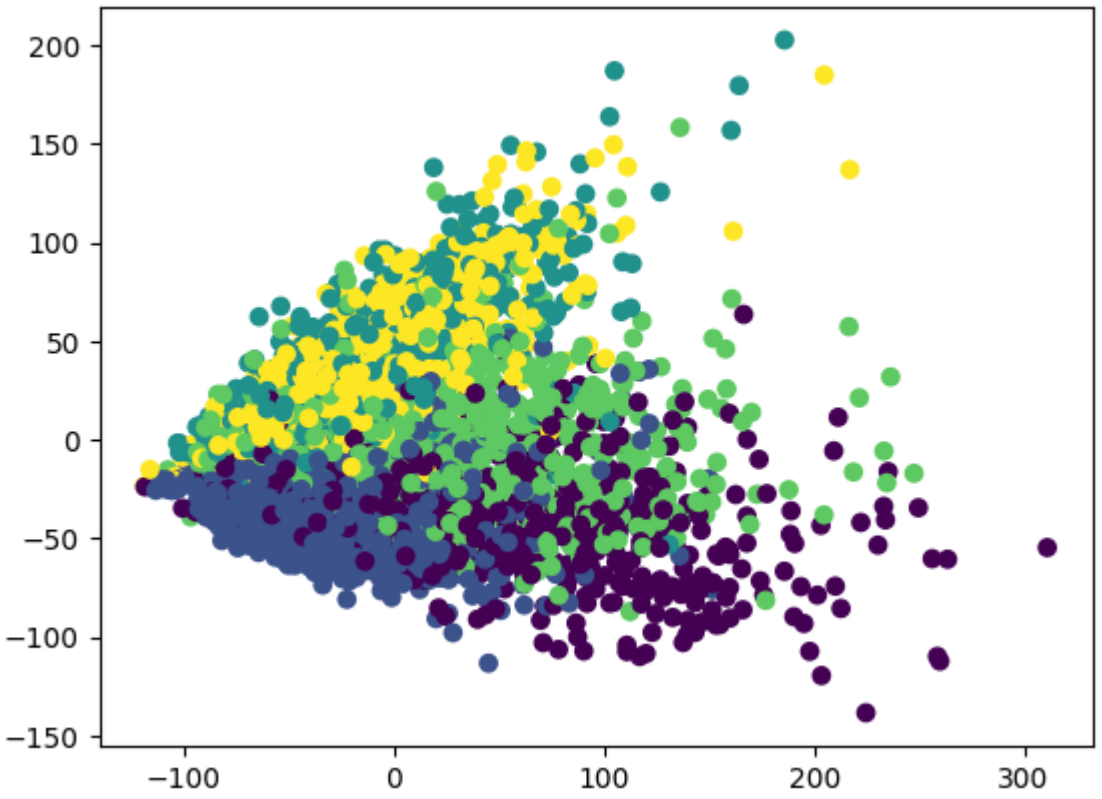
ANSWER-22

The extracted features from a VGG network are dense. In a sparse representation, most elements of the vector are zero, indicating the absence of a term in a particular document. VGG features are dense vectors, meaning that a significant portion of the elements in the vector has non-zero values. VGG extracts features at multiple levels, capturing intricate details, textures, and structures.

In contrast in text data with methods like TF-IDF, the focus is often on extracting the most relevant words or terms. Given the vast vocabulary in text, creating dense vectors for every word would be inefficient. TF-IDF is made to be a Sparse representation with the aim to emphasize essential terms that differentiate documents. They assign importance to specific words while ignoring the majority to save computational resources.

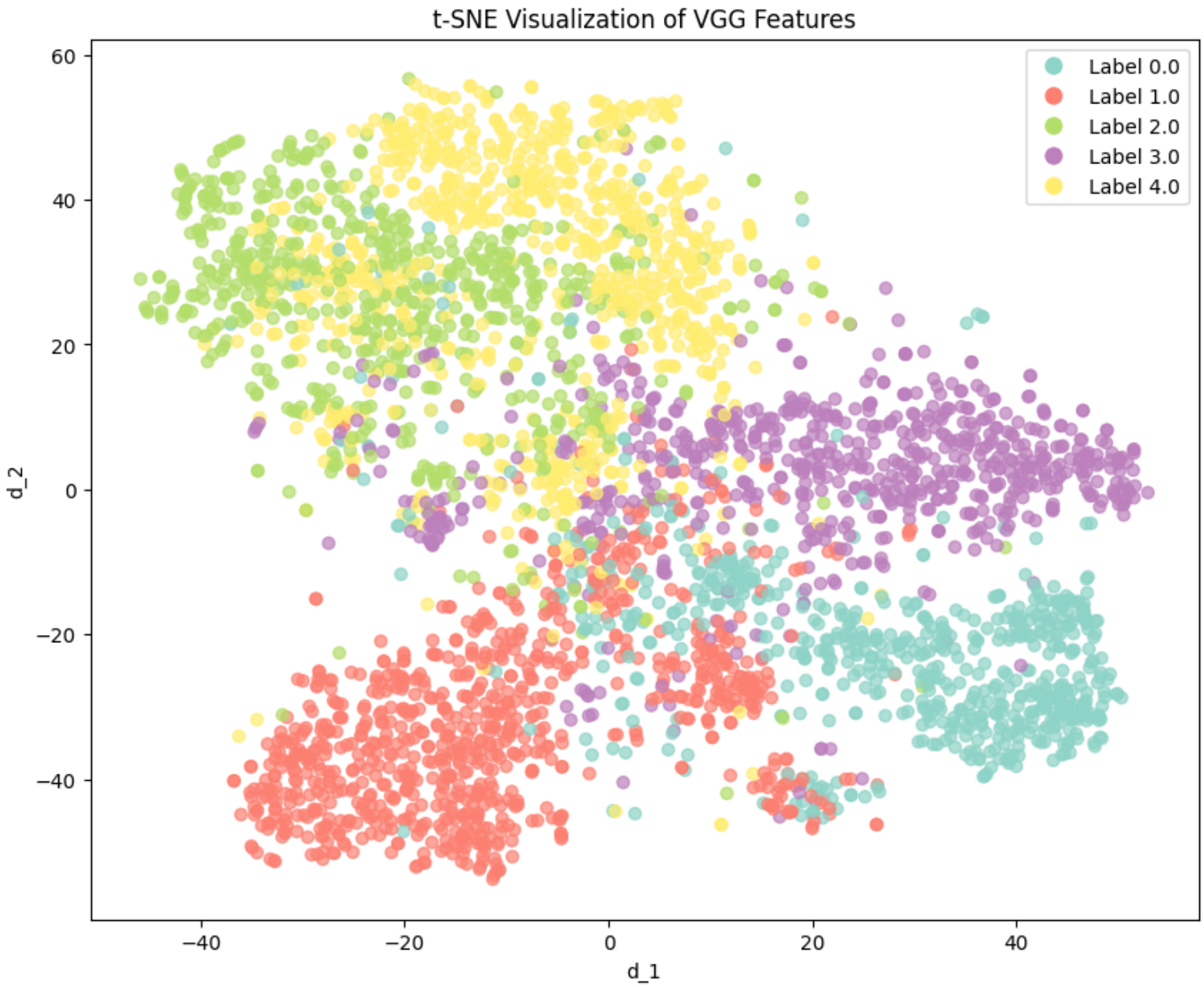
In [ ]:
```
f_pca = PCA(n_components=2).fit_transform(f_all)
plt.scatter(*f_pca.T, c=y_all)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x79c37a52aef0>
```



In [ ]:

```
from sklearn.manifold import TSNE
from matplotlib.lines import Line2D

tsne = TSNE(n_components=2, random_state=42)
mapped_features = tsne.fit_transform(f_all)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(mapped_features[:, 0], mapped_features[:, 1], c=y_all, cmap='Set3',alp
ha=0.7)
legend_colors = scatter.to_rgba(np.unique(y_all))
legend_elements = [Line2D([0], [0], marker='o', color='w', label=f'Label {label}', markerfac
ecolor=color, markersize=10) for label, color in zip(np.unique(y_all), legend_colors)]
plt.legend(handles=legend_elements, loc='upper right')
plt.title('t-SNE Visualization of VGG Features')
plt.xlabel('d_1')
plt.ylabel('d_2')
plt.show()
```



ANSWER-23

We are able to visualize the clusters more distinctively in the second plot which is t-SNE compared to PCA. This is because t-SNE has the ability to capture non-linear relationships and preserve local structures in the data. t-SNE is also well-suited for high-dimensional data like ours. Also, t-SNE is effective at preserving the local relationships between data points. It tends to

maintain the distances between neighboring points which is important for visualizing clusters. PCA in constrast focuses more on global variance and may not preserve local structures well.

Autoencoder

In [ ]:

```
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_compone
nts))

    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 [ ]:
```
X_em =Autoencoder(2).fit_transform(f_all)
plt.scatter(*X_em.T, c=y_all)
```

Param-grid search

In [ ]:

```
ground_truth_labels= y_all
# Define hyperparameter grids
dim_reduction_methods = ['Autoencoder','TruncatedSVD', 'UMAP']
clustering_methods = [ 'HDBSCAN','KMeans', 'Agglomerative']
k_values = [10, 20, 50]  # Hyperparameter values for KMeans
min_cluster_sizes = [100,200,500,1000]  # Hyperparameter values for HDBSCAN min_cluster_size
min_samples_values = [5, 30,60,100,250,500,1000]  # Hyperparameter values for HDBSCAN min_sa
mples
n_clusters_values = [20]  # Hyperparameter values for Agglomerative n_clusters

best_score = 0.0
best_config = None
r=50 # rank = 50 for dim reduction
# Iterate over both dimensionality reduction and clustering methods simultaneously


for dim_reduction_method in dim_reduction_methods:
    for clustering_method in clustering_methods:
            print("Running {} with {}".format(dim_reduction_method, clustering_method))
            if dim_reduction_method == 'TruncatedSVD':
                reduce_dim = TruncatedSVD(n_components=50)
            elif dim_reduction_method == 'UMAP':
                reduce_dim = umap.UMAP(n_components=50, metric= 'cosine')
            else:
                reduce_dim = Autoencoder(n_components=50)
            features = reduce_dim.fit_transform(f_all)
            if clustering_method == 'KMeans':
                for k in k_values:
                    cluster_model = KMeans(n_clusters=k)
                    labels = cluster_model.fit_predict(features)  # Assuming 'data' is your
input data
                    score = adjusted_rand_score(ground_truth_labels, labels)
                    if score > best_score:
                        best_score = score
                        best_config = (dim_reduction_method, clustering_method, 50, k)
            elif clustering_method == 'HDBSCAN':
                for min_cluster_size in min_cluster_sizes:
                    for min_samples in min_samples_values:
                        cluster_model = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size, m
in_samples=min_samples)
                        labels = cluster_model.fit_predict(features)  # Assuming 'data' is y
our input data
                        score = adjusted_rand_score(ground_truth_labels, labels)
                        if score > best_score:
                            best_score = score
                            best_config = (dim_reduction_method, clustering_method, 50, min_
```

```
cluster_size, min_samples)
            elif clustering_method == 'Agglomerative':
                for n_clusters in n_clusters_values:
                    cluster_model = AgglomerativeClustering(n_clusters=n_clusters)
                    labels = cluster_model.fit_predict(features)  # Assuming 'data' is your
input data
                    score = adjusted_rand_score(ground_truth_labels, labels)
                    if score > best_score:
                        best_score = score
                        best_config = (dim_reduction_method, clustering_method, 50, n_cluste
rs)

print(f"Best Configuration: {best_config}, Best V-Measure Score: {best_score}")
```

```
Running Autoencoder with HDBSCAN
```

```
100%|██████████| 100/100 [00:18<00:00,  5.31it/s]
```

```
Running Autoencoder with KMeans
```

```
100%|██████████| 100/100 [00:18<00:00,  5.30it/s]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Running Autoencoder with Agglomerative
```

```
100%|██████████| 100/100 [00:18<00:00,  5.26it/s]
```

```
Running TruncatedSVD with HDBSCAN
Running TruncatedSVD with KMeans
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Running TruncatedSVD with Agglomerative
Running UMAP with HDBSCAN
Running UMAP with KMeans
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expli
citly to suppress the warning
  warnings.warn(
```

```
Running UMAP with Agglomerative
Best Configuration: ('UMAP', 'KMeans', 50, 10), Best V-Measure Score: 0.4255204864081218
```

In [ ]:

```
print("------------------- BEST DIM_RED + CLUSTERING ------------------- \n")

print("Best dimension reduction : " , best_config[0], "   |  Rank = ", 50)
print("Best clustering          : ", best_config[1], " |  Number of clusters = ", best_confi
g[3])
print("-----------------------------------------------------------------")
```

```
------------------- BEST DIM_RED + CLUSTERING -------------------

Best dimension reduction :  UMAP    |  Rank =  50
Best clustering          :  KMeans  |  Number of clusters =  10
-----------------------------------------------------------------
```

ANSWER-24

We note from above cell:

Best Dimension Reduction:

- Technique: UMAP
- Rank: 50

Best Clustering:

- Method: KMeans
- Number of Clusters: 10

Best Adjusted Rand Index Score:

- ARI = 0.426

MLP Classifier

In [ ]:

```
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()
                outputs = self.model(X_)
                loss = criterion(outputs, 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')

        self.model.eval()
```

```
        with torch.no_grad():
            outputs = self.model(X_test)
            _, predicted = torch.max(outputs, 1)

        accuracy = (predicted == y_test).sum().item() / len(y_test)
        return accuracy
        ###################################
```

MLP on original feature set

In [ ]:
```
num_features = f_all.shape[1]  # Assuming the features are in the second dimension

f_train, f_test, y_train, y_test = train_test_split(f_all, y_all, test_size=0.2, random_stat
e=42)

mlp = MLP(num_features=num_features)
mlp.train(f_train, y_train)
accuracy = mlp.eval(f_test, y_test)  # Replace f_test and y_test with your test data
print(f"Test Accuracy (Original VGG features): {accuracy}")
```

```
Test Accuracy: 0.9046321525885559
```

MLP on reduced feature set

In [ ]:
```
reduce_dim = umap.UMAP(n_components=50, metric= 'cosine')
reduced_f_all = reduce_dim.fit_transform(f_all)
reduced_f_train, reduced_f_test, y_train, y_test = train_test_split(reduced_f_all, y_all, te
st_size=0.2, random_state=42)

mlp = MLP(num_features=50)
mlp.train(reduced_f_train, y_train)
accuracy_red = mlp.eval(reduced_f_test, y_test)  # Replace f_test and y_test with your test
data
print(f"Test Accuracy (Reduced VGG features, n_components=50): {accuracy_red}")
```

```
Test Accuracy (Reduced VGG features, n_components=50): 0.8174386920980926
```

ANSWER-25

We use UMAP to reduce the number of features down to 50.

Test Accuracy:

- Original: 0.9046
- Reduced VGG features (n_components=50): 0.8174

The reduction in accuracy from 0.9046 (original) to 0.8174 (reduced VGG features with n_components=50) indicates a decrease in classification accuracy. Although the reduction is noticeable, it may not be considered highly significant.

The success in classification, even with reduced features, aligns with the clustering results obtained in Question 24. The chosen combination of dimension reduction (UMAP with Rank=50) and clustering (KMeans with 10 clusters) was found to be the best based on the custom score of 0.557. This suggests that despite the dimensionality reduction, the features still capture essential information for successful classification, and the clustering results further support the meaningful structure within the data.

Image dataset: https://www.kaggle.com/datasets/hlrhegemony/pokemon-image-dataset?resource=download (https://www.kaggle.com/datasets/hlrhegemony/pokemon-image-dataset?resource=download)

Metainfo csv: https://github.com/lgreski/pokemonData/blob/master/Pokemon.csv (https://github.com/lgreski/pokemonData/blob/master/Pokemon.csv)

You can speed up model inference on colab by changing Hardware accelerator in runtime type to any GPU option

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
!pip install datasets transformers numpy pandas Pillow matplotlib
!pip install torch tqdm scipy
!pip install git+https://github.com/openai/CLIP.git
!pip install plotly umap-learn
```

```
Collecting datasets
  Downloading datasets-2.17.0-py3-none-any.whl (536 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.6/536.6 kB 8.6 MB/s
eta 0:00:00
Requirement already satisfied: transformers in /usr/local/lib/python3.
10/dist-packages (4.35.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist
-packages (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dis
t-packages (1.5.3)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dis
t-packages (9.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.1
0/dist-packages (3.7.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/d
ist-packages (from datasets) (3.13.1)
Collecting pyarrow>=12.0.0 (from datasets)
  Downloading pyarrow-15.0.0-cp310-cp310-manylinux_2_28_x86_64.whl (3
8.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.3/38.3 MB 40.4 MB/s e
ta 0:00:00
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python
3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 16.1 MB/s
eta 0:00:00
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/pyth
on3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.
10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dis
t-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 20.5 MB/s
eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /
usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/di
st-packages (from datasets) (3.9.3)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/l
ib/python3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/
dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.1
0/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers) (2023.12.25)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/li
b/python3.10/dist-packages (from transformers) (0.15.1)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/py
thon3.10/dist-packages (from transformers) (0.4.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/li
b/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.
10/dist-packages (from pandas) (2023.4)
```

```
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pyth
on3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.
10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from matplotlib) (4.48.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pyth
on3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/pyth
on3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python
3.10/dist-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/p
ython3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python
3.10/dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/l
ib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/loca
l/lib/python3.10/dist-packages (from huggingface-hub>=0.19.4->dataset
s) (4.9.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/d
ist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/
lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.
10/dist-packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/py
thon3.10/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/py
thon3.10/dist-packages (from requests>=2.19.0->datasets) (2024.2.2)
Installing collected packages: pyarrow, dill, multiprocess, datasets
  Attempting uninstall: pyarrow
    Found existing installation: pyarrow 10.0.1
    Uninstalling pyarrow-10.0.1:
      Successfully uninstalled pyarrow-10.0.1
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of t
he following dependency conflicts.
ibis-framework 7.1.0 requires pyarrow<15,>=2, but you have pyarrow 15.
0.0 which is incompatible.
Successfully installed datasets-2.17.0 dill-0.3.8 multiprocess-0.70.16
pyarrow-15.0.0
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist
-packages (2.1.0+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (4.66.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist
-packages (1.11.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/d
ist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/pyt
hon3.10/dist-packages (from torch) (4.9.0)
```

```
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist
-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/d
ist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dis
t-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dis
t-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python
3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/li
b/python3.10/dist-packages (from scipy) (1.23.5)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pytho
n3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.
10/dist-packages (from sympy->torch) (1.3.0)
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-578
poyri
  Running command git clone --filter=blob:none --quiet https://github.
com/openai/CLIP.git /tmp/pip-req-build-578poyri
  Resolved https://github.com/openai/CLIP.git to commit a1d071733d7111
c9c014f024669f959182114e33
  Preparing metadata (setup.py) ... done
Collecting ftfy (from clip==1.0)
  Downloading ftfy-6.1.3-py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.4/53.4 kB 1.8 MB/s et
a 0:00:00
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist
-packages (from clip==1.0) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from clip==1.0) (4.66.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist
-packages (from clip==1.0) (2.1.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.1
0/dist-packages (from clip==1.0) (0.16.0+cu121)
Requirement already satisfied: wcwidth<0.3.0,>=0.2.12 in /usr/local/li
b/python3.10/dist-packages (from ftfy->clip==1.0) (0.2.13)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/d
ist-packages (from torch->clip==1.0) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/pyt
hon3.10/dist-packages (from torch->clip==1.0) (4.9.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist
-packages (from torch->clip==1.0) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/d
ist-packages (from torch->clip==1.0) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dis
t-packages (from torch->clip==1.0) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dis
t-packages (from torch->clip==1.0) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python
3.10/dist-packages (from torch->clip==1.0) (2.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist
-packages (from torchvision->clip==1.0) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/d
ist-packages (from torchvision->clip==1.0) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/li
```

```
b/python3.10/dist-packages (from torchvision->clip==1.0) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pytho
n3.10/dist-packages (from jinja2->torch->clip==1.0) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/
lib/python3.10/dist-packages (from requests->torchvision->clip==1.0)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.
10/dist-packages (from requests->torchvision->clip==1.0) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/py
thon3.10/dist-packages (from requests->torchvision->clip==1.0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/py
thon3.10/dist-packages (from requests->torchvision->clip==1.0) (2024.
2.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.
10/dist-packages (from sympy->torch->clip==1.0) (1.3.0)
Building wheels for collected packages: clip
  Building wheel for clip (setup.py) ... done
  Created wheel for clip: filename=clip-1.0-py3-none-any.whl size=1369
497 sha256=a939806bea7c0f6ee5fd6ae9be1e97965d95485736c0dd2bcb9d224ea53
57561
  Stored in directory: /tmp/pip-ephem-wheel-cache-ctocem20/wheels/da/2
b/4c/d6691fa9597aac8bb85d2ac13b112deb897d5b50f5ad9a37e4
Successfully built clip
Installing collected packages: ftfy, clip
Successfully installed clip-1.0 ftfy-6.1.3
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dis
t-packages (5.15.0)
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.9/90.9 kB 2.7 MB/s et
a 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/pytho
n3.10/dist-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/
dist-packages (from plotly) (23.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.1
0/dist-packages (from umap-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.
10/dist-packages (from umap-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/py
thon3.10/dist-packages (from umap-learn) (1.2.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python
3.10/dist-packages (from umap-learn) (0.58.1)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.8/55.8 kB 7.9 MB/s et
a 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (4.66.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/loca
l/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn) (0.41.
1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.
10/dist-packages (from pynndescent>=0.5->umap-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/
python3.10/dist-packages (from scikit-learn>=0.22->umap-learn) (3.2.0)
```

```
Building wheels for collected packages: umap-learn
  Building wheel for umap-learn (setup.py) ... done
  Created wheel for umap-learn: filename=umap_learn-0.5.5-py3-none-an
y.whl size=86832 sha256=d3535021a20ac45863c30fc36c3f2674d20b8b1887c785
642d510873ce3c8742
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a
3b431db59b806a10da736612ebbc66c1bcc5
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
```

In [3]:
```
!pip install  opendatasets
```

```
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from opendatasets) (4.66.1)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dis
t-packages (from opendatasets) (1.5.16)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist
-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/
dist-packages (from kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/di
st-packages (from kaggle->opendatasets) (2024.2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/pytho
n3.10/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/d
ist-packages (from kaggle->opendatasets) (2.31.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python
3.10/dist-packages (from kaggle->opendatasets) (8.0.3)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/di
st-packages (from kaggle->opendatasets) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dis
t-packages (from kaggle->opendatasets) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.
10/dist-packages (from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/p
ython3.10/dist-packages (from python-slugify->kaggle->opendatasets)
(1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/
lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.
3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.
10/dist-packages (from requests->kaggle->opendatasets) (3.6)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

In [14]:
```
import opendatasets as od
od.download('https://www.kaggle.com/datasets/hlrhegemony/pokemon-image-dataset')
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: itamii
Your Kaggle Key: ··········
Downloading pokemon-image-dataset.zip to ./pokemon-image-dataset

100%|██████████| 57.9M/57.9M [00:02<00:00, 28.7MB/s]
```

In [4]:
```
from datasets import load_dataset
from transformers import CLIPProcessor, CLIPModel
import numpy as np
import pandas as pd
from glob import glob
from PIL import Image
import matplotlib.pyplot as plt
import clip
import torch
from tqdm import tqdm
from scipy.special import softmax
import plotly.express as px
import plotly.graph_objects as go
from sklearn.manifold import TSNE
from PIL import Image
```

In [5]:
```
%cd '/content/drive/MyDrive/219/Project2'
```

```
/content/drive/MyDrive/219/Project2
```

In [6]:
```
def construct_pokedex(csv_path='Pokemon.csv', image_dir='pokemon-image
-dataset/images', type_to_load=None):
    pokedex = pd.read_csv(csv_path)
    image_paths = []

    for pokemon_name in pokedex["Name"]:
        imgs = glob(f"{image_dir}/{pokemon_name}/0.jpg")
        if len(imgs) > 0:
            image_paths.append(imgs[0])
        else:
            image_paths.append(None)

    pokedex["image_path"] = image_paths
    pokedex = pokedex[pokedex["image_path"].notna()].reset_index(drop=
True)
    ids, id_counts = np.unique(pokedex["ID"], return_counts=True)
    ids, id_counts = np.array(ids), np.array(id_counts)
    keep_ids = ids[id_counts == 1]

    pokedex = pokedex[pokedex["ID"].isin(keep_ids)].reset_index(drop=T
rue)
    pokedex["Type2"] = pokedex["Type2"].str.strip()
    if type_to_load is not None:
        pokedex = pokedex[pokedex["Type1"].isin(type_to_load)].reset_i
ndex(drop=True)
    return pokedex

def load_clip_model():
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model, preprocess = clip.load("ViT-L/14", device=device)
    return model, preprocess, device

def clip_inference_image(model, preprocess, image_paths, device):
    image_embeddings = []
    with torch.no_grad():
        for img_path in tqdm(image_paths):
            img = Image.open(img_path)
            img_preprocessed = preprocess(img).unsqueeze(0).to(device)
            image_embedding = model.encode_image(img_preprocessed).det
ach().cpu().numpy()
            image_embeddings += [image_embedding]

    image_embeddings = np.concatenate(image_embeddings, axis=0)
    image_embeddings /= np.linalg.norm(image_embeddings, axis=-1, keep
dims=True)
    return image_embeddings

def clip_inference_text(model, preprocess, texts, device):
    with torch.no_grad():
        text_embeddings = model.encode_text(clip.tokenize(texts).to(de
vice)).detach().cpu().numpy()
    text_embeddings /= np.linalg.norm(text_embeddings, axis=-1, keepdi
ms=True)
    return text_embeddings

def compute_similarity_text_to_image(image_embeddings, text_embedding
s):
```

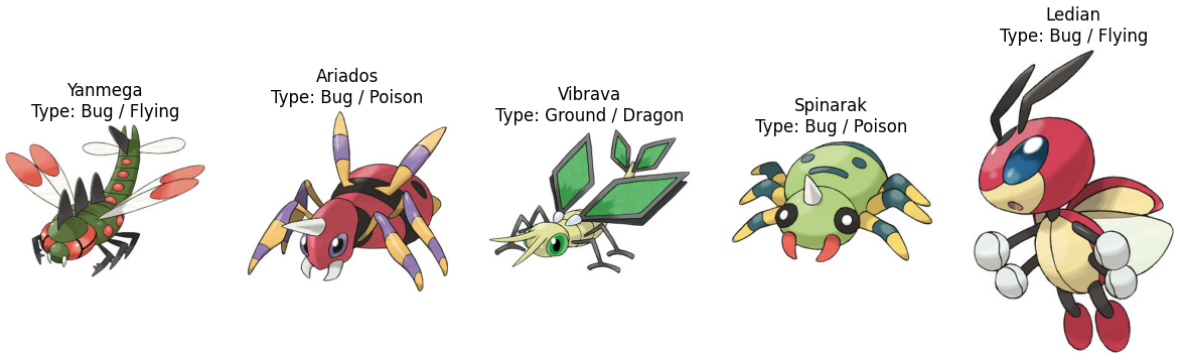
```
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.
T), axis=-1)
    return similarity

def compute_similarity_image_to_text(image_embeddings, text_embedding
s):
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.
T), axis=0)
    return similarity

def umap_projection(image_embeddings, n_neighbors=15, min_dist=0.1, me
tric='cosine'):
    distance_matrix = np.zeros((image_embeddings.shape[0], image_embed
dings.shape[0]))
    for i in range(image_embeddings.shape[0]):
        for j in range(image_embeddings.shape[0]):
            if i == j:
                distance_matrix[i, j] = 1
            else:
                distance_matrix[i, j] = np.dot(image_embeddings[i], im
age_embeddings[j])
    distance_matrix = 1 - distance_matrix
    reducer = TSNE(n_components=2, metric="precomputed", init="rando
m", random_state=42)
    visualization_data = reducer.fit_transform(distance_matrix)
    return visualization_data
```

In [30]:
```
def get_top_relevant_pokemon(pokedex, model, preprocess, device, quer
y, num_pokemon=5):
    # Inference CLIP model on text query
    text_embeddings = clip_inference_text(model, preprocess, [query],
device)[0]
    image_paths = pokedex["image_path"].tolist()
    image_embeddings = clip_inference_image(model, preprocess, image_p
aths, device)
    similarity_scores = compute_similarity_text_to_image(image_embeddi
ngs, text_embeddings)
    top_indices = np.argsort(similarity_scores)[-num_pokemon:][::-1]
    relevant_pokemon = pokedex.loc[top_indices, ["Name", "Type1", "Typ
e2","image_path"]]

    return relevant_pokemon

def plot_relevant_pokemon(relevant_pokemon, title):
    fig, axes = plt.subplots(1, len(relevant_pokemon), figsize=(15,
5))
    fig.suptitle(title)

    for i, (_, row) in enumerate(relevant_pokemon.iterrows()):
        ax = axes[i] if len(relevant_pokemon) > 1 else axes
        img_path = row["image_path"]
        img = Image.open(img_path)
        ax.imshow(img)
        ax.axis("off")
        ax.set_title(f"{row['Name']}\nType: {row['Type1']}{' / ' + row
['Type2'] if pd.notna(row['Type2']) else ''}")

    plt.show()
```

In [20]:
```
pokedex = construct_pokedex()
## Load CLIP model
model, preprocess, device = load_clip_model()

bug_query = "Pokemon type: Bug"
fire_query = "Pokemon type: Fire"
grass_query = "Pokemon type: Grass"

bug_relevant_pokemon = get_top_relevant_pokemon(pokedex, model, prepro
cess, device, bug_query, num_pokemon=5)
fire_relevant_pokemon = get_top_relevant_pokemon(pokedex, model, prepr
ocess, device, fire_query, num_pokemon=5)
grass_relevant_pokemon = get_top_relevant_pokemon(pokedex, model, prep
rocess, device, grass_query, num_pokemon=5)

# Plot the relevant Pokemon
plot_relevant_pokemon(bug_relevant_pokemon, title="Bug Type Pokemon")
plot_relevant_pokemon(fire_relevant_pokemon, title="Fire Type Pokemo
n")
plot_relevant_pokemon(grass_relevant_pokemon, title="Grass Type Pokemo
n")
```

```
100%|██████████| 753/753 [00:20<00:00, 36.67it/s]
100%|██████████| 753/753 [00:18<00:00, 39.85it/s]
100%|██████████| 753/753 [00:20<00:00, 36.90it/s]
```

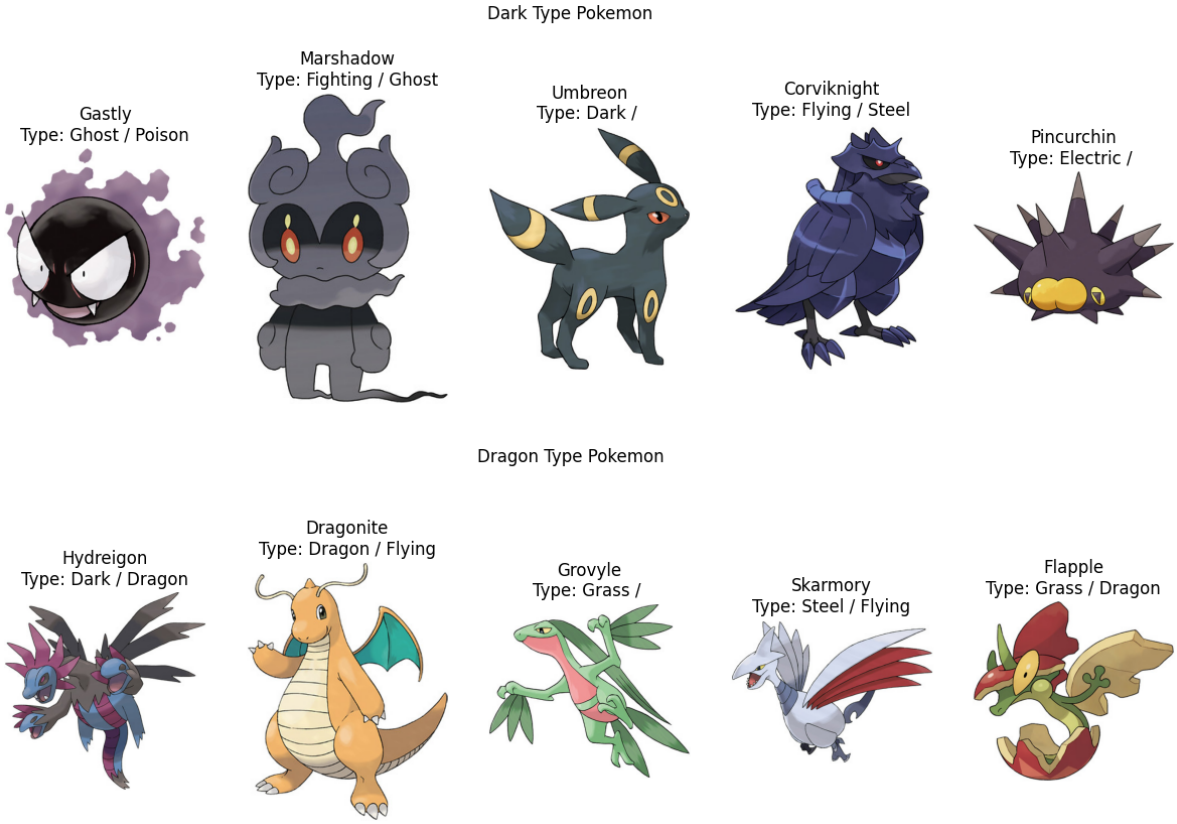
Bug Type Pokemon



Fire Type Pokemon

Grass Type Pokemon

In [32]:
```
dark_query = "Pokemon type: Dark"
dragon_query = "Pokemon type: Dragon"

dark_relevant_pokemon = get_top_relevant_pokemon(pokedex, model, prepr
ocess, device, dark_query, num_pokemon=5)
dragon_relevant_pokemon = get_top_relevant_pokemon(pokedex, model, pre
process, device, dragon_query, num_pokemon=5)

plot_relevant_pokemon(dark_relevant_pokemon, title="Dark Type Pokemo
n")
plot_relevant_pokemon(dragon_relevant_pokemon, title="Dragon Type Poke
mon")
```



ANSWER-26

We note that the queries function well for Bug, Fire and Grass pokemons (Error rates : 1/5, 0,0) respectively. For Dark type, it makes 4 errors out of 5 and for dragon it is 2 errors out of 5. We note that these errors arise due to overlap of certain features with other dragon types hence making the decision to pick a type more confusing. Dark for instance shares visual appearances similr to ghost and electric, and since CLIP looks at similarity between the text and the image, it could confuse one type for another if they visually have similar patterns. Take the dragon type for instance, "Grovyle" and "Skarmory" Pokemons in the image above look visually like dragons and hence get misclassified as Dragon types when in reality they are Grass and Steel types.

Part-2

In [43]:
```
pokemon_types = pokedex["Type1"].unique()
formatted_types = [f"Pokemon type: {t}" for t in pokemon_types]
```

In [44]:
```
formatted_types
```

Out[44]:
```
['Pokemon type: Grass',
 'Pokemon type: Fire',
 'Pokemon type: Water',
 'Pokemon type: Bug',
 'Pokemon type: Normal',
 'Pokemon type: Poison',
 'Pokemon type: Fairy',
 'Pokemon type: Fighting',
 'Pokemon type: Psychic',
 'Pokemon type: Electric',
 'Pokemon type: Ghost',
 'Pokemon type: Rock',
 'Pokemon type: Ground',
 'Pokemon type: Ice',
 'Pokemon type: Dragon',
 'Pokemon type: Dark',
 'Pokemon type: Steel',
 'Pokemon type: Flying']
```

ANSWER-27

Images and top-5 types are as below in the cell output

In [12]:
```
import random

def randomly_select_pokemon(pokedex, num_pokemon=10):
    selected_indices = random.sample(range(len(pokedex)), num_pokemon)
    selected_pokemon = pokedex.loc[selected_indices, ["Name", "Type1",
"Type2", "image_path"]]
    return selected_pokemon

def get_most_relevant_types(pokemon, pokedex, model, preprocess, devic
e):
    image_path = pokemon["image_path"]
    image_embeddings = clip_inference_image(model, preprocess, [image_
path], device)
    all_types_init = pokedex["Type1"].unique()
    all_types = [f"Pokemon type: {t}" for t in all_types_init]
    all_types = np.array(all_types)
    all_types_embeddings = clip_inference_text(model, preprocess, all_
types, device)
    similarity_scores = compute_similarity_text_to_image(image_embeddi
ngs, all_types_embeddings)
    similarity_scores_flat = similarity_scores.flatten()
    top_indices = np.argsort(similarity_scores_flat)[-5:][::-1]
    top_types = all_types[top_indices]
    top_probabilities = similarity_scores_flat[top_indices]

    return top_types.tolist(), top_probabilities.tolist()

def plot_selected_pokemon(selected_pokemon, pokedex, model, preproces
s, device):
    for _, pokemon in selected_pokemon.iterrows():
        top_types, top_probabilities = get_most_relevant_types(pokemo
n, pokedex, model, preprocess, device)
        formatted_top_types = [f"Pokemon type: {t}" for t in top_type
s]
        plt.figure(figsize=(10, 6))
        plt.imshow(Image.open(pokemon["image_path"]))
        plt.title(f"{pokemon['Name']}\nType: {pokemon['Type1']}{' / '
+ pokemon['Type2'] if pd.notna(pokemon['Type2']) else ''}")
        plt.axis("off")
        plt.show()

        print("Top 5 Most Relevant Types:")
        for t, p in zip(top_types, top_probabilities):
            print(f"  - {t}: Probability {p:.2f}")
        print("\n" + "="*50 + "\n")



selected_pokemon = randomly_select_pokemon(pokedex, num_pokemon=10)
plot_selected_pokemon(selected_pokemon, pokedex, model, preprocess, de
vice)
```

```
100%|██████████| 1/1 [00:00<00:00,  1.59it/s]
```


```
Top 5 Most Relevant Types:
  - Pokemon type: Psychic: Probability 0.66
  - Pokemon type: Dark: Probability 0.12
  - Pokemon type: Normal: Probability 0.09
  - Pokemon type: Rock: Probability 0.04
  - Pokemon type: Ghost: Probability 0.02

================================================

100%|██████████| 1/1 [00:01<00:00,  1.69s/it]
```

Cherrim
Type: Grass /

```
Top 5 Most Relevant Types:
  - Pokemon type: Poison: Probability 0.83
  - Pokemon type: Dark: Probability 0.07
  - Pokemon type: Psychic: Probability 0.03
  - Pokemon type: Normal: Probability 0.03
  - Pokemon type: Rock: Probability 0.01

================================================


100%|██████████| 1/1 [00:00<00:00,  1.45it/s]
```

Lickilicky
Type: Normal /

```
Top 5 Most Relevant Types:
  - Pokemon type: Normal: Probability 0.40
  - Pokemon type: Psychic: Probability 0.11
  - Pokemon type: Rock: Probability 0.11
  - Pokemon type: Electric: Probability 0.07
  - Pokemon type: Ice: Probability 0.06

================================================


100%|██████████| 1/1 [00:00<00:00,  1.56it/s]
```

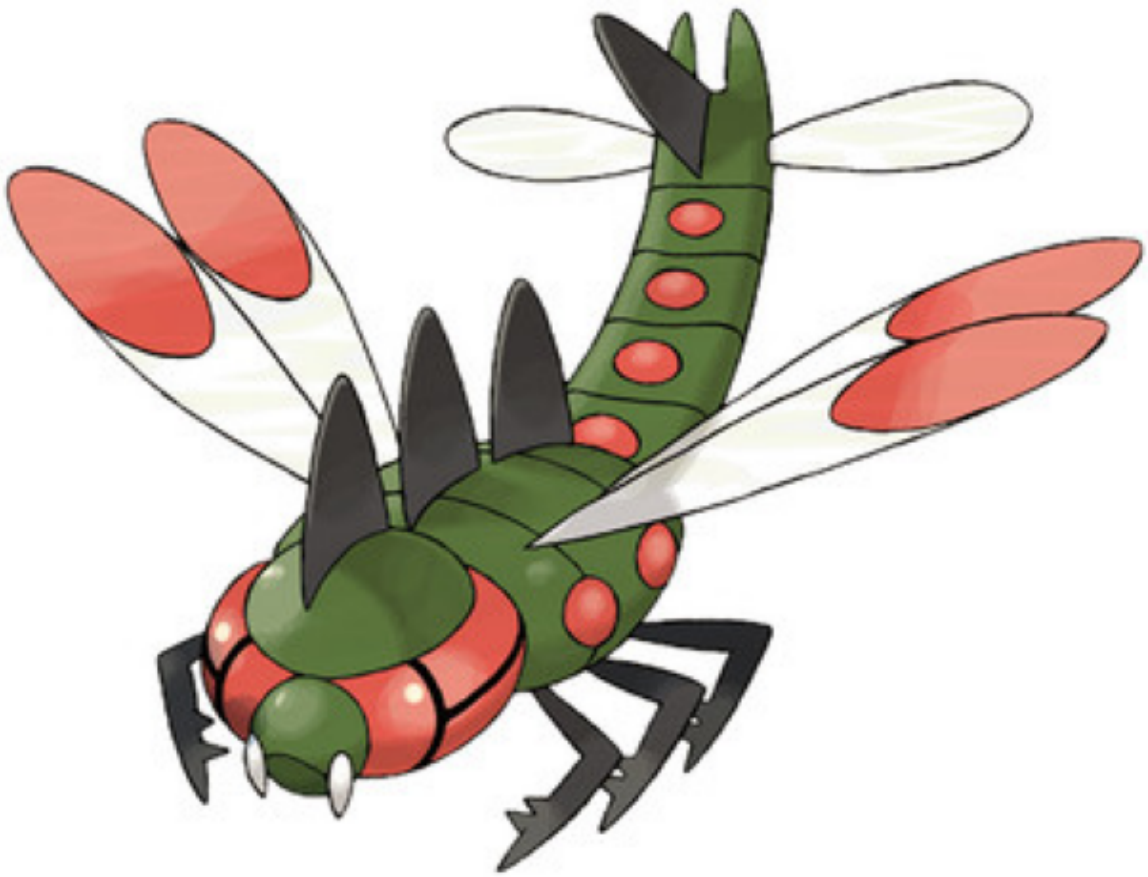
Elekid
Type: Electric /

```
Top 5 Most Relevant Types:
  - Pokemon type: Electric: Probability 0.87
  - Pokemon type: Rock: Probability 0.05
  - Pokemon type: Ground: Probability 0.02
  - Pokemon type: Normal: Probability 0.01
  - Pokemon type: Psychic: Probability 0.01

==================================================


100%|██████████| 1/1 [00:00<00:00,  1.47it/s]
```

Croagunk
Type: Poison / Fighting

```
Top 5 Most Relevant Types:
  - Pokemon type: Normal: Probability 0.20
  - Pokemon type: Electric: Probability 0.18
  - Pokemon type: Dark: Probability 0.15
  - Pokemon type: Ground: Probability 0.10
  - Pokemon type: Rock: Probability 0.10

================================================

100%|██████████| 1/1 [00:00<00:00,  1.43it/s]
```

Yanmega
Type: Bug / Flying



```
Top 5 Most Relevant Types:
  - Pokemon type: Bug: Probability 0.92
  - Pokemon type: Flying: Probability 0.04
  - Pokemon type: Normal: Probability 0.01
  - Pokemon type: Poison: Probability 0.01
  - Pokemon type: Rock: Probability 0.00

==================================================

100%|██████████| 1/1 [00:00<00:00,  1.38it/s]
```

Fennekin
Type: Fire /

```
Top 5 Most Relevant Types:
  - Pokemon type: Fire: Probability 0.79
  - Pokemon type: Psychic: Probability 0.05
  - Pokemon type: Normal: Probability 0.04
  - Pokemon type: Electric: Probability 0.02
  - Pokemon type: Fairy: Probability 0.02

==================================================

100%|██████████| 1/1 [00:01<00:00,  1.17s/it]
```

Roselia
Type: Grass / Poison

```
Top 5 Most Relevant Types:
  - Pokemon type: Fairy: Probability 0.83
  - Pokemon type: Poison: Probability 0.09
  - Pokemon type: Psychic: Probability 0.03
  - Pokemon type: Ice: Probability 0.01
  - Pokemon type: Electric: Probability 0.01

================================================


100%|██████████| 1/1 [00:00<00:00,  1.49it/s]
```

Volcarona
Type: Bug / Fire

```
Top 5 Most Relevant Types:
  - Pokemon type: Bug: Probability 0.66
  - Pokemon type: Flying: Probability 0.07
  - Pokemon type: Dark: Probability 0.06
  - Pokemon type: Fairy: Probability 0.05
  - Pokemon type: Rock: Probability 0.05

================================================


100%|██████████| 1/1 [00:00<00:00,  1.38it/s]
```

Poliwag
Type: Water /

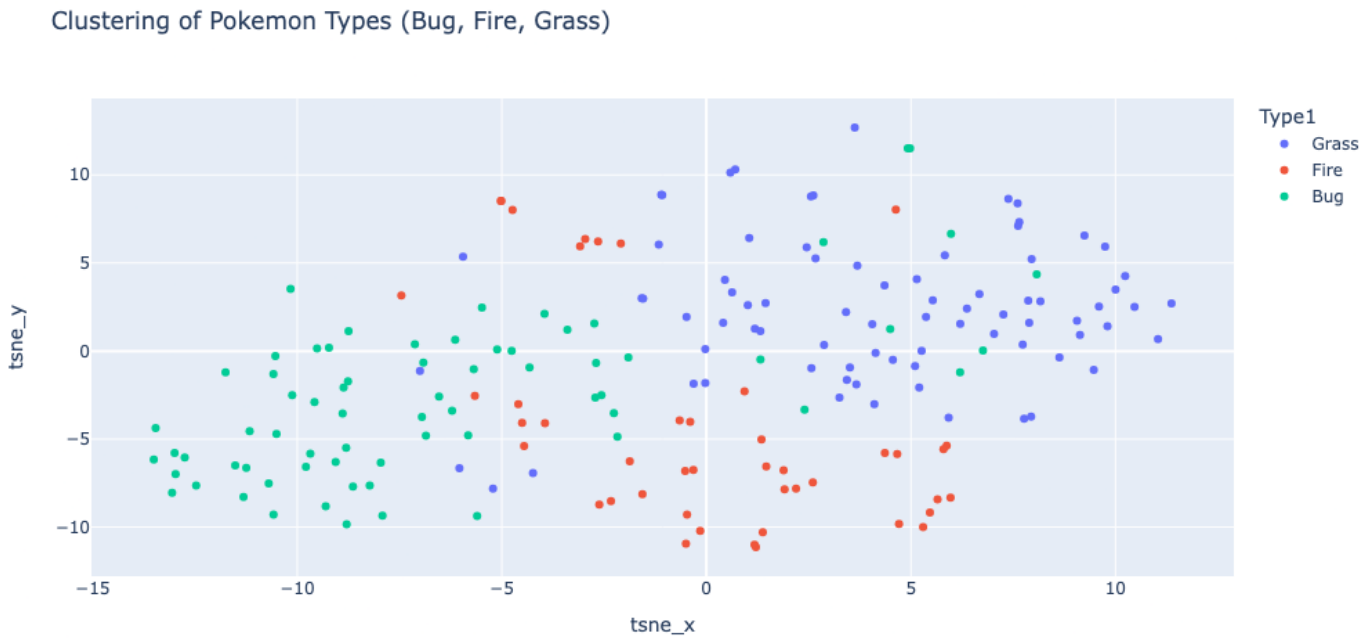
```
Top 5 Most Relevant Types:
  - Pokemon type: Water: Probability 0.90
  - Pokemon type: Psychic: Probability 0.04
  - Pokemon type: Ice: Probability 0.02
  - Pokemon type: Electric: Probability 0.01
  - Pokemon type: Normal: Probability 0.01

================================================
```

Part-3

In [28]:
```
selected_types = ["Bug", "Fire", "Grass"]
selected_data = pokedex[pokedex["Type1"].isin(selected_types)]
selected_image_paths = selected_data["image_path"].tolist()
selected_image_embeddings = clip_inference_image(model, preprocess, se
lected_image_paths, device)
visualization_data = umap_projection(selected_image_embeddings)

selected_data["tsne_x"] = visualization_data[:, 0]
selected_data["tsne_y"] = visualization_data[:, 1]
fig = px.scatter(
    selected_data,
    x='tsne_x',
    y='tsne_y',
    color='Type1',
    hover_name='Name',
    title='Clustering of Pokemon Types (Bug, Fire, Grass)',
)

fig.show()
```

```
<ipython-input-28-26fa331aafb6>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  selected_data["tsne_x"] = visualization_data[:, 0]
<ipython-input-28-26fa331aafb6>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  selected_data["tsne_y"] = visualization_data[:, 1]
```

ANSWER-28



When we use t-SNE to visualize how Pokemon images are grouped in a space, we see some patterns related to their types like Bug in bottom left, Fire in bottom , and Grass in top right. It's like trying to see if Pokemon of the same type tend to stick together in clusters. However, because Pokemon types can be diverse and some may look similar, the groups might not be perfectly separated and have outliers. We can see some Pokemon of different types appearing close to each other. So, while it gives us a general idea, it might not capture all the complexities and variations in how Pokemon look. It's kind of like trying to group similar-looking Pokemon together, but there can be exceptions and overlap between different types.

In [ ]: