Part 1

Clustering with Sparse Text Representations

```
!pip install regex
!pip install nltk
!pip install sklearn
!pip install umap-learn[plot]
!pip install holoviews
!pip install -U ipykernel
!pip install ClusterEnsembles
```

```
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/d
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
Requirement already satisfied: umap-learn[plot] in /usr/local/lib/python3.10/
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.1
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.10/
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: datashader in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: bokeh in /usr/local/lib/python3.10/dist-packag
```

```
Requirement already satisfied: bokeh in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: holoviews in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: colorcet in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: contourpy>=1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: packaging>=16.8 in /usr/local/lib/python3.10/d
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: xyzservices>=2021.09.1 in /usr/local/lib/pytho
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pyct>=0.4.4 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: dask in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: multipledispatch in /usr/local/lib/python3.10/
Requirement already satisfied: param in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: pyviz-comms>=0.7.4 in /usr/local/lib/python3.1
```

```
import numpy as np
import sklearn
import nltk, string
import matplotlib.pyplot as plt

import itertools
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
def plot_mat(mat, xticklabels = None, yticklabels = None, pic_fname = None, size=
             colorbar = True, grid = 'k', xlabel = None, ylabel = None, title = N
    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, edgecolo

    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_facec
            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, **k

    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()

from sklearn.datasets import fetch_20newsgroups

# get dataset
categories = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardwa
                  'rec.autos', 'rec.motorcycles','rec.sport.baseball', 'rec.sport
newsgroups = fetch_20newsgroups(subset = 'train',categories=categories, remove=('

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

# count vectorizer on corpus
tf_vectorizer = CountVectorizer(min_df = 3, stop_words='english')
newsgroups_vectorized = tf_vectorizer.fit_transform(newsgroups.data)

# count vector to TF-IDF
transformer = TfidfTransformer()
newsgroups_tfidf = transformer.fit_transform(newsgroups_vectorized)

print('TF-IDF Dimensions: ', newsgroups_tfidf.shape)
```

```
    TF-IDF Dimensions:  (4732, 17131)
```

Question 1

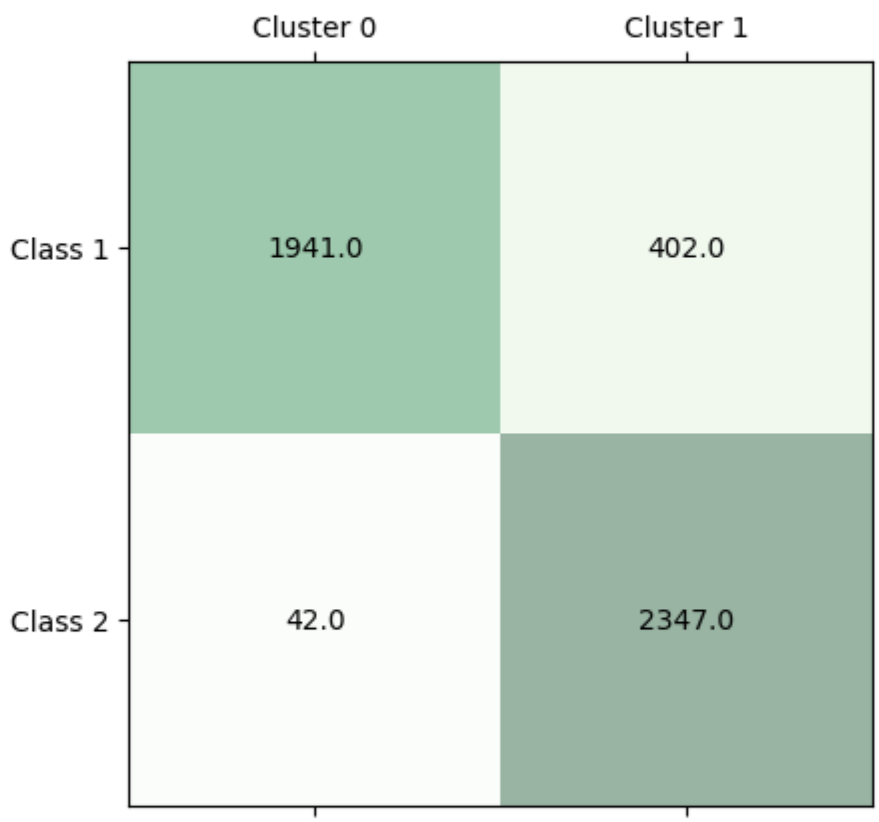
Dimensions of the TF-IDF matrix is (4732, 17131)

```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import contingency_matrix
import matplotlib.pyplot as plt
import numpy as np

# Get clusters
kmeans = KMeans(random_state=0, n_clusters=2, max_iter=1000, n_init=30).fit(newsg

label_kmeans = []
for label in newsgroups.target:
    if label in [0, 1, 2, 3]:
        label_kmeans.append(0)
    else:
        label_kmeans.append(1)
contingency_table = contingency_matrix(label_kmeans, kmeans.labels_)
print('Contingency Table:  ', '\n', contingency_table)
```

```
Contingency Table:
 [[1941  402]
 [  42 2347]]
```

Question 2

```
from sklearn import metrics

# plot contingency matrix
plt.matshow(contingency_table, cmap=plt.cm.Greens, alpha=0.4)
for (i, j), z in np.ndenumerate(contingency_table):
    plt.text(j, i, '{:0.1f}'.format(z), ha='center', va='center')
plt.xticks(range(2), ['Cluster 0', 'Cluster 1'])
plt.yticks(range(2), ['Class 1', 'Class 2'])
plt.show()
```



*** Q2 Answer: ***

The presented contingency table illustrates the outcomes of clustering. From the prominent diagonal pattern, we deduce an association between group 1 and category 2, as well as group 0 and category 1. Considering that we configured Kmeans with 2 clusters, aligning with the 2 categories in our data, the contingency matrix is expected to be square. Any discrepancy between the cluster count set in Kmeans and the data's category count would result in a non-square contingency matrix.

Question 3

```
from sklearn.metrics import cluster

print("Homogeneity score: %0.3f" % cluster.homogeneity_score(label_kmeans, kmeans
print("Completeness score: %0.3f" % cluster.completeness_score(label_kmeans, kmea
print("V-measure score: %0.3f" % cluster.v_measure_score(label_kmeans, kmeans.lab
print("Adjusted Rand Index: %0.3f" % cluster.adjusted_rand_score(label_kmeans, km
print("Adjusted mutual information score: %0.3f" % cluster.adjusted_mutual_info_s
```

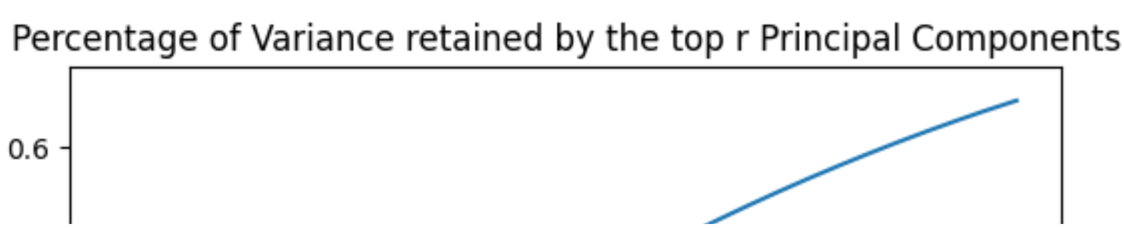
```
Homogeneity score: 0.589
Completeness score: 0.601
V-measure score: 0.595
Adjusted Rand Index: 0.660
Adjusted mutual information score: 0.595
```

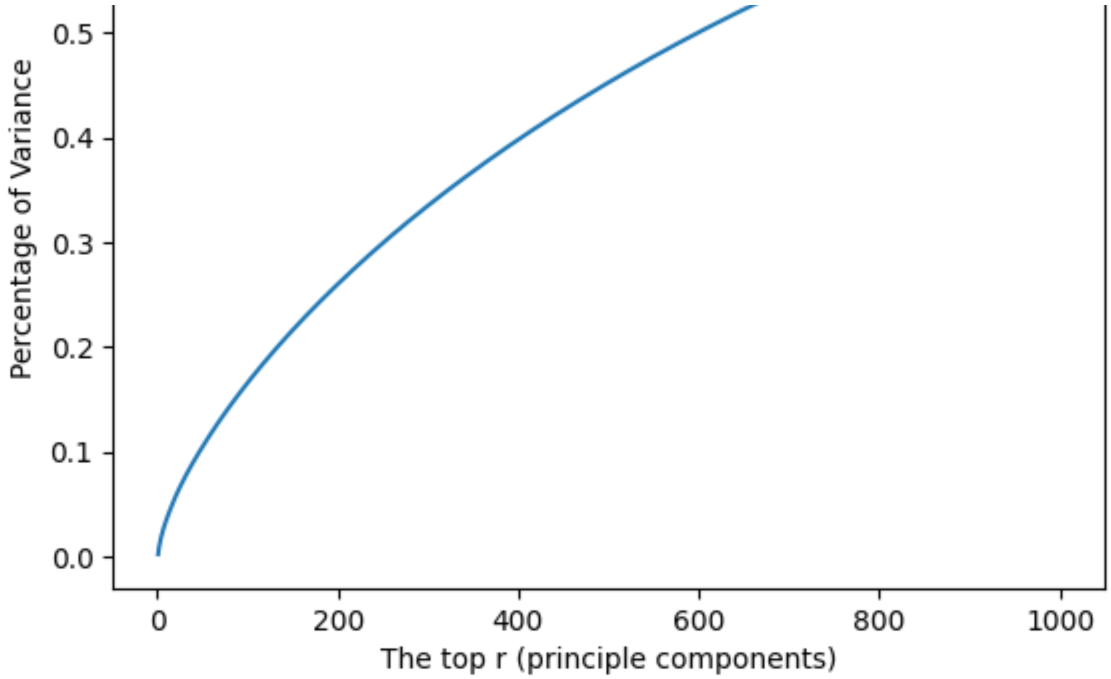
Clustering with Dense Text Representations

1. Generate dense representations for better K-Means Clustering

Question 4

```
from sklearn.decomposition import TruncatedSVD

# get principle components
svd = TruncatedSVD(n_components=1000, random_state=0)
newsgroups_lsi = svd.fit_transform(newsgroups_tfidf)

# get explained variance ratio
x = np.linspace(1, 1000, 1000)
ratio = svd.explained_variance_ratio_.cumsum()

# plot explained variance ratio
plt.plot(x, ratio)
plt.ylabel('Percentage of Variance')
plt.xlabel('The top r (principle components)')
plt.title('Percentage of Variance retained by the top r Principal Components')
plt.show()
```

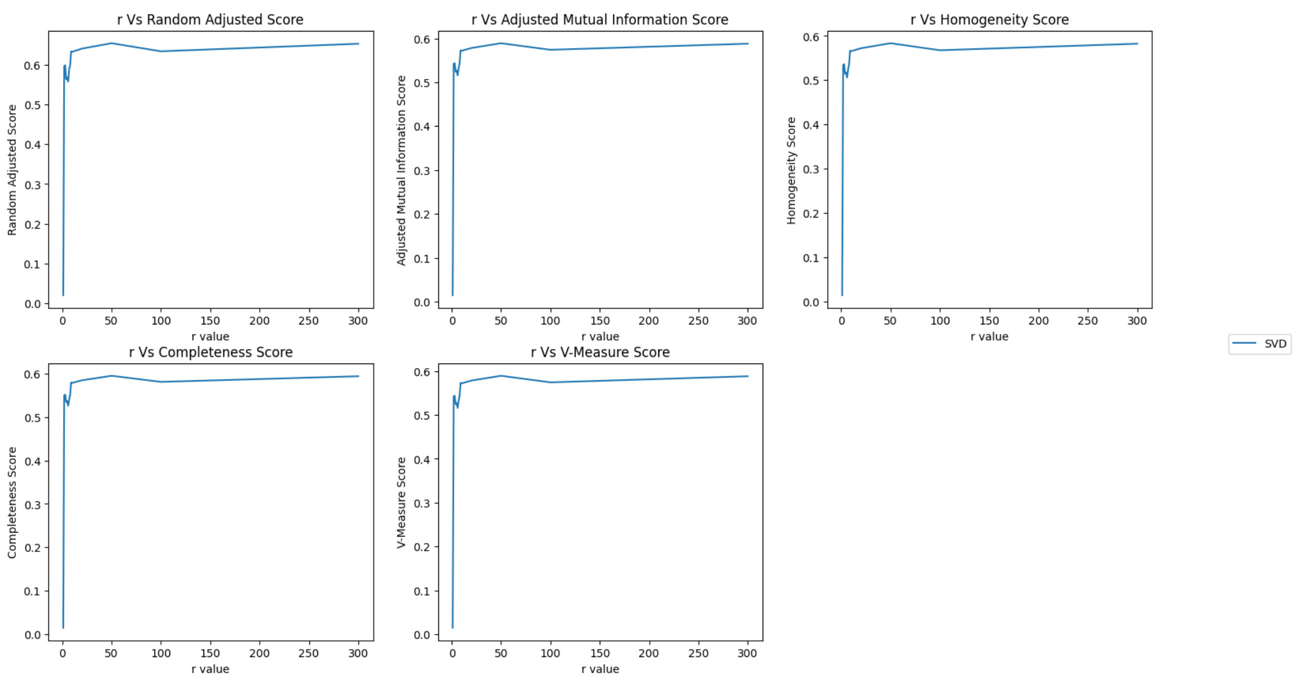




Question 5

```
from sklearn.metrics import adjusted_rand_score, adjusted_mutual_info_score, homc
import statistics

def calculate_svd_scores(r, k, X, y):
    svd_adj_rand_score = []
    svd_adj_mutual_score = []
    svd_hom_score = []
    svd_comp_score = []
    svd_v_score = []

    for dim in r:
        svd = TruncatedSVD(n_components=dim, random_state=0)
        truncated_svd = svd.fit_transform(X)
        kmeans = KMeans(random_state=0, n_clusters=k, max_iter=1000, n_init=30)
        kmeans.fit(truncated_svd)

        svd_adj_rand_score.append(adjusted_rand_score(y, kmeans.labels_))
        svd_adj_mutual_score.append(adjusted_mutual_info_score(y, kmeans.labels_)
        svd_hom_score.append(homogeneity_score(y, kmeans.labels_))
        svd_comp_score.append(completeness_score(y, kmeans.labels_))
        svd_v_score.append(v_measure_score(y, kmeans.labels_))

    return svd_adj_rand_score, svd_adj_mutual_score, svd_hom_score, svd_comp_scor

def plot_metrics_vs_r(r, svd_adj_rand_score, svd_adj_mutual_score, svd_hom_score,
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

    def plot(ax, data, label, title, xlabel, ylabel):
```

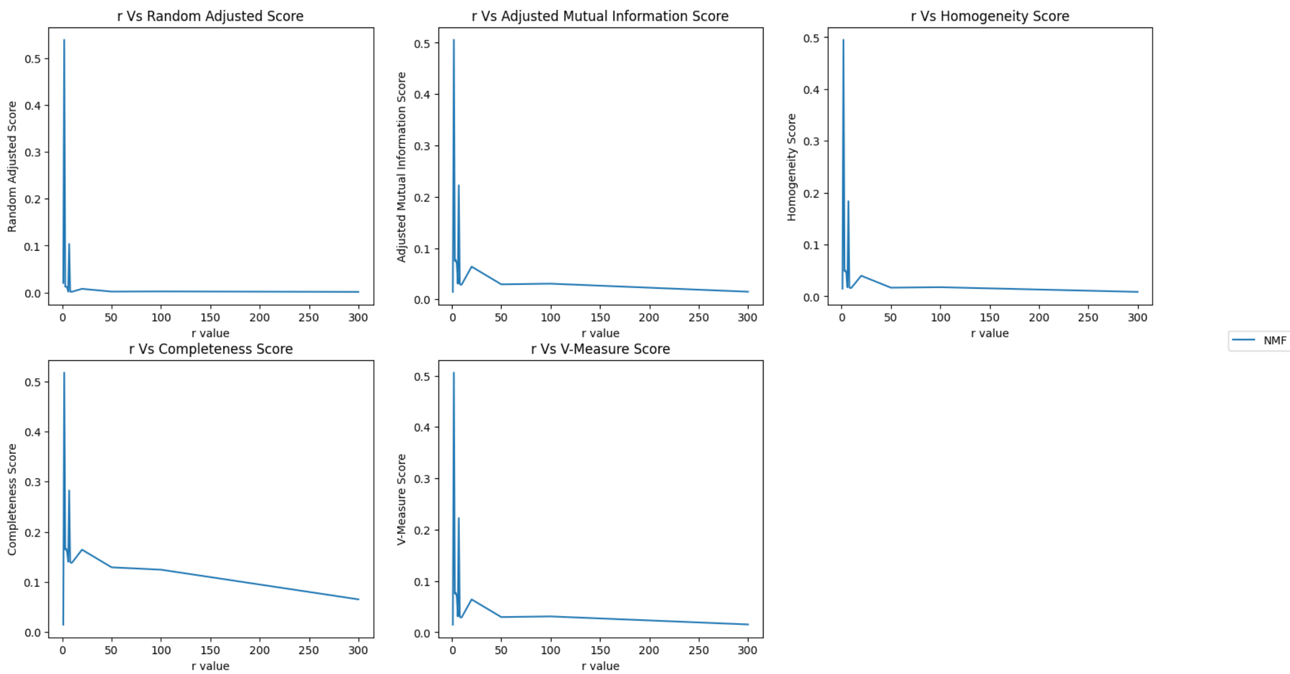
```
        ax.plot(r, data, label=label)
        ax.set_title(title)
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)

    plot(axes[0, 0], svd_adj_rand_score, 'SVD', 'r Vs Random Adjusted Score', 'r
    plot(axes[0, 1], svd_adj_mutual_score, 'SVD', 'r Vs Adjusted Mutual Informati
    plot(axes[0, 2], svd_hom_score, 'SVD', 'r Vs Homogeneity Score', 'r value', '
    plot(axes[1, 0], svd_comp_score, 'SVD', 'r Vs Completeness Score', 'r value',
    plot(axes[1, 1], svd_v_score, 'SVD', 'r Vs V-Measure Score', 'r value', 'V-Me

    axes[1, 2].axis('off')
    fig.legend(['SVD'], loc='center right')

    plt.show()

def find_best_r_value(scores):
    argmaxes = [i.index(max(i)) for i in scores]
    best_r_ind = round(statistics.mode(argmaxes))
    return best_r_ind

r = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50, 100, 300]
k = 2

svd_scores = calculate_svd_scores(r, k,  newsgroups_tfidf, label_kmeans)

svd_adj_rand_score, svd_adj_mutual_score, svd_hom_score, svd_comp_score, svd_v_sc

plot_metrics_vs_r(r, svd_adj_rand_score, svd_adj_mutual_score, svd_hom_score, svc

best_svd_r_value = r[find_best_r_value(svd_scores)]
print('Best SVD r value:', best_svd_r_value)
```



```
Best SVD r value: 50
```

```
from sklearn.decomposition import NMF
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
import statistics

def calculate_nmf_scores(r_values, k, X, y):
    adj_rand_score = []
    adj_mutual_score = []
    hom_score = []
    comp_score = []
    v_score = []

    for dim in r_values:
        nmf = NMF(n_components=dim, init='random', random_state=0, max_iter=500)
        trunc_nmf = nmf.fit_transform(X)
        kmeans = KMeans(random_state=0, n_clusters=k, max_iter=1000, n_init=30)
        kmeans.fit(trunc_nmf)
        adj_rand_score.append(metrics.adjusted_rand_score(y, kmeans.labels_))
        adj_mutual_score.append(metrics.adjusted_mutual_info_score(y, kmeans.labe
        hom_score.append(metrics.homogeneity_score(y, kmeans.labels_))
        comp_score.append(metrics.completeness_score(y, kmeans.labels_))
        v_score.append(metrics.v_measure_score(y, kmeans.labels_))

    return adj_rand_score, adj_mutual_score, hom_score, comp_score, v_score

def plot_nmf_scores(r_values, scores):
    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))
    metrics_names = ['Random Adjusted', 'Adjusted Mutual Information', 'Homogenei

    for i, metric in enumerate(scores):
        row, col = divmod(i, 3)
        axes[row, col].plot(r_values, metric, label='NMF')
        axes[row, col].set_title(f'r Vs {metrics_names[i]} Score')
        axes[row, col].set_xlabel('r value')
        axes[row, col].set_ylabel(f'{metrics_names[i]} Score')

    axes[1, 2].axis('off')
    fig.legend(['NMF'], loc='center right')
```

```
    plt.show()

def find_best_r_value(scores):
    argmaxes = [metric.index(max(metric)) for metric in scores]
    best_r_ind = round(statistics.mode(argmaxes))
    return best_r_ind

# Assuming you have r, newsgroups_tfidf, label_kmeans defined
r_values = r
k_clusters = 2

nmf_scores = calculate_nmf_scores(r_values, k_clusters, newsgroups_tfidf, label_k
nmf_adj_rand_score, nmf_mutual_score, nmf_hom_score, nmf_comp_score, nmf_v_score

plot_nmf_scores(r_values, [nmf_adj_rand_score, nmf_mutual_score, nmf_hom_score, r

nmf_score = [nmf_adj_rand_score, nmf_mutual_score, nmf_hom_score, nmf_comp_score,
best_nmf_r_value = find_best_r_value(nmf_score)

print('Best NMF r value:', r[best_nmf_r_value])
```




```
Best NMF r value: 2
```

Q5 Answer:

A good choice of r for SVD is 50. A good choice of r for NMF is 2.

Question 6

While dimensionality reduction helps to deal with noisy data and shorten the running time of the algorithm, it may also lead to loss of information, including noise. Thus, as the dimensionality reduction parameter r increases, the accuracy of KMeans clustering may decrease because we truncate too much data. However, as r increases, we may observe that the clustering score initially increases and then decreases. The initial increase indicates that we have struck a good balance between scores and truncated data. An eventual decrease may indicate that as the truncated data increases, more noise is introduced, leading to inaccurate KMeans clustering and lower scores. Thus, we can observe a non-monotonic behavior of the measurements as r increases.

Question 7

```
def print_average_metrics(method, hom_score, comp_score, v_score, adj_rand_score,
    print(f"\n{method} Metrics:")
    print("Homogeneity: ", np.mean(hom_score))
    print("Completeness: ", np.mean(comp_score))
    print("V-measure: ", np.mean(v_score))
    print("Adjusted Rand-Index: ", np.mean(adj_rand_score))
    print("Adjusted Mutual Information: ", np.mean(adj_mutual_score))

print_average_metrics("SVD", svd_hom_score, svd_comp_score, svd_v_score, svd_adj_
print_average_metrics("NMF", nmf_hom_score, nmf_comp_score, nmf_v_score, nmf_adj_
```

```
SVD Metrics:
Homogeneity:  0.5086295760481007
Completeness:  0.5226573391383235
```

```
V-measure:  0.5155424912/1773
Adjusted Rand-Index:  0.567182526171996
Adjusted Mutual Information:  0.5154676533294708

NMF Metrics:
Homogeneity:  0.0704770686887141
Completeness:  0.16756615651774417
V-measure:  0.08748731407893089
Adjusted Rand-Index:  0.051330192251747724
Adjusted Mutual Information:  0.08726492696038733
```

Q7 Answer

Both SVD and NMF metrics, on average, are worse than those computed in Question 3. However, SVD performs relatively better and is closer to the metrics from Question 3 compared to NMF.

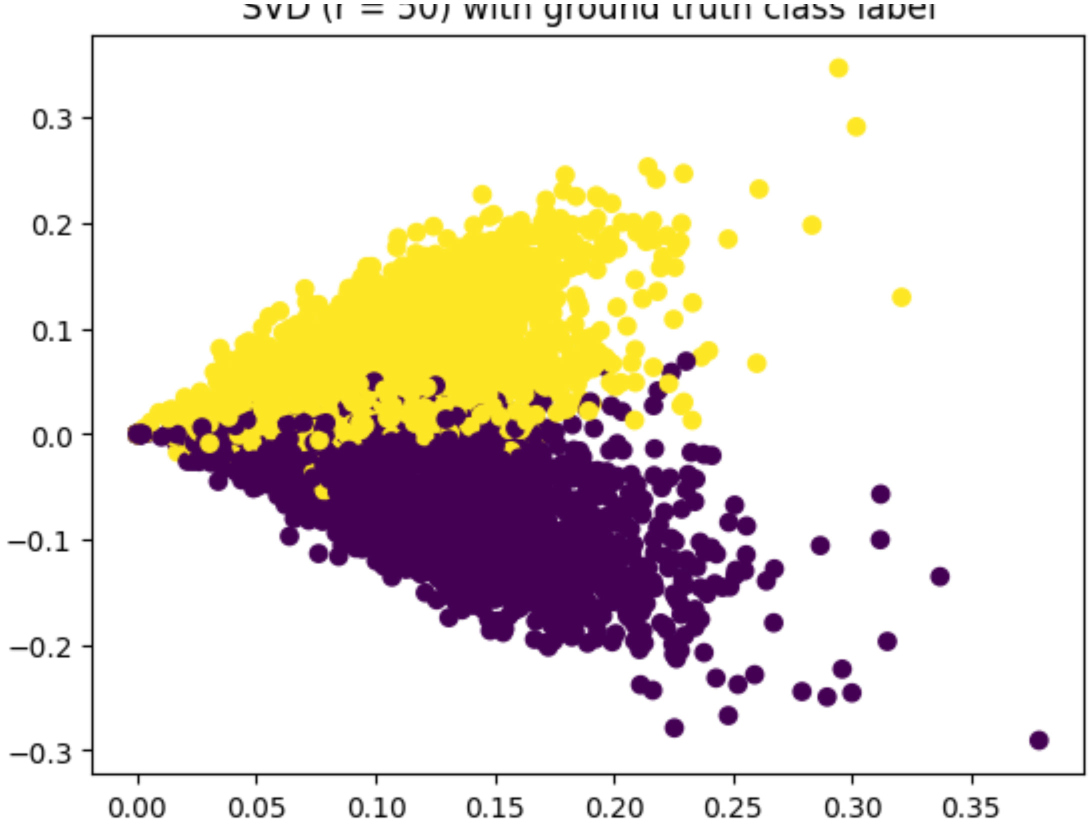
2. Visualize the clusters

Question 8

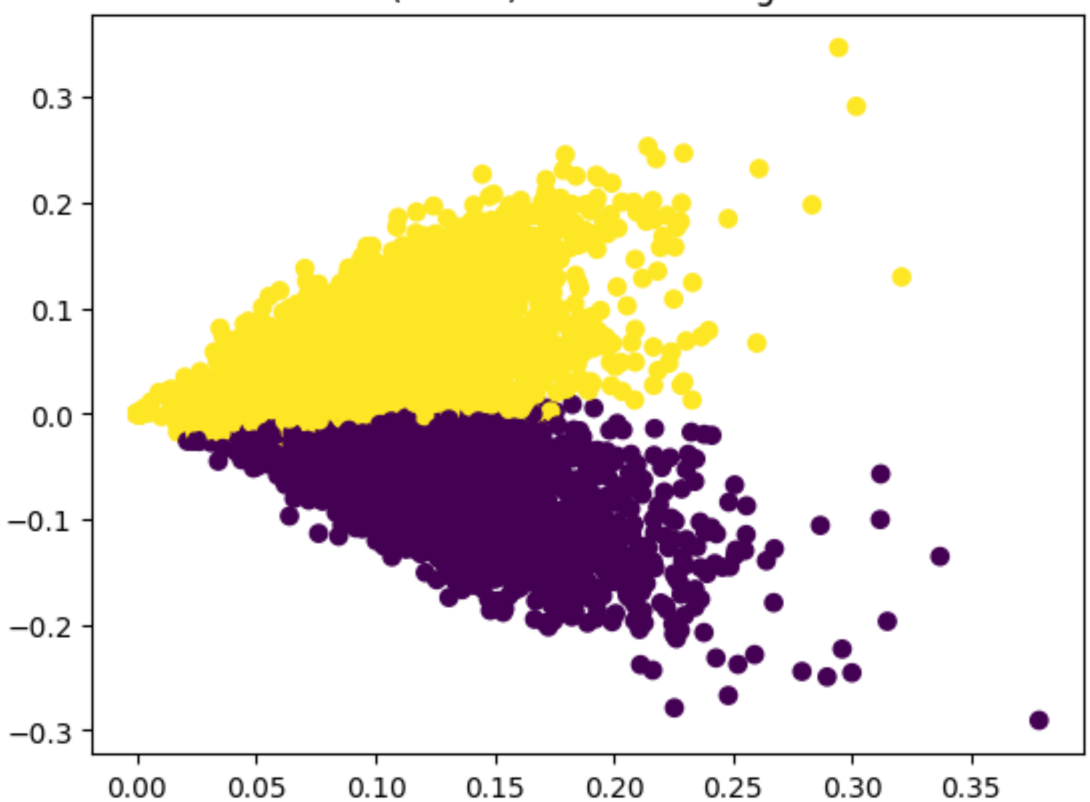
```
from sklearn.decomposition import TruncatedSVD, NMF
import matplotlib.pyplot as plt

def perform_svd(data, components=50, random_state=42):
    svd_model = TruncatedSVD(n_components=components, random_state=random_state)
    svd_transformed = svd_model.fit_transform(data)
    return svd_transformed

def perform_nmf(data, components=2, random_state=0):
    nmf_model = NMF(n_components=components, init='random', random_state=random_s
    nmf_transformed = nmf_model.fit_transform(data)
    return nmf_transformed

def plot_scatter(transformed_data, labels, title):
    plt.scatter(transformed_data[:, 0], transformed_data[:, 1], c=labels)
    plt.title(title)
    plt.show()


svd_transformed_data = perform_svd(newsgroups_tfidf)
nmf_transformed_data = perform_nmf(newsgroups_tfidf)

plot_scatter(svd_transformed_data, label_kmeans, "SVD (r = 50) with ground truth
plot_scatter(svd_transformed_data, kmeans.labels_, "SVD (r = 50) with clustering
plot_scatter(nmf_transformed_data, label_kmeans, "NMF (r = 2) with ground truth c
plot_scatter(nmf_transformed_data, kmeans.labels_, "NMF (r = 2) with clustering l
```

SVD (r = 50) with ground truth class label

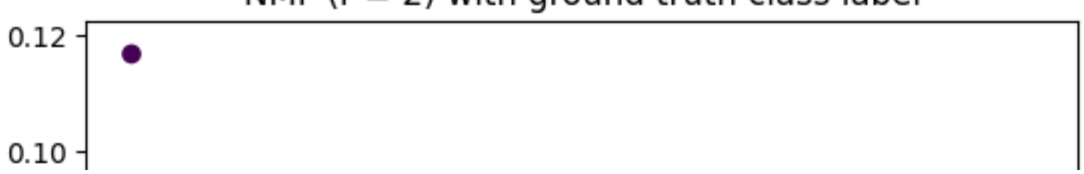
SVD (r = 50) with ground truth class label



SVD (r = 50) with clustering label



NMF (r = 2) with ground truth class label



Question 9

The aforementioned graphs reveal a striking similarity between the clustered labels and the actual group labels. Nevertheless, the genuine group labels exhibit a greater level of overlap, a nuance not distinctly evident in the labeling graphs generated by NMF and SVD, where labeling boundaries are more clearly defined. The data portrays a triangular distribution rather than a spherical one, with centroids of individual labels closely positioned. Simultaneously, outliers are present at a considerable distance from the primary clusters. Given that K-Means clustering assumes a spherical data distribution, this non-spherical data distribution poses a suboptimal scenario.

3. Clustering of the Entire 20 Classes

Question 10

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
import pandas as pd

# Load the dataset
news_dataset = fetch_20newsgroups(subset = 'all',shuffle = True, random_state = 0

def load_and_transform_dataset(min_df=3):

    # Create CountVectorizer and TfidfTransformer
    vectorizer = CountVectorizer(stop_words="english", min_df=min_df)
    transformer = TfidfTransformer(use_idf=True)

    # Transform the text data
    word_count_matrix = vectorizer.fit_transform(news_dataset.data)
    tfidf_matrix = transformer.fit_transform(word_count_matrix)
    tfidf_array = tfidf_matrix.toarray()

    # Get feature names from CountVectorizer
    feature_names = vectorizer.get_feature_names_out()

    # Create a DataFrame with the transformed data
    tfidf_dataframe = pd.DataFrame(data=tfidf_array, columns=feature_names)

    return tfidf_dataframe

# Example usage
tfidf_dataframe = load_and_transform_dataset()
```

```
print(tfidf_dataframe.shape)
```

```
(18846, 45365)
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.metrics import homogeneity_score, completeness_score, v_measure_scor

def calculate_best_svd_score(r_values, data, kmeans_clusters, target_labels):
    best_score_svd = 0
    best_r_svd = 0

    for r in r_values:
        print(r)
        svd_model = TruncatedSVD(n_components=r, random_state=42)
        svd_features = svd_model.fit_transform(data)
        kmeans_clusters.fit(svd_features)

        hs = homogeneity_score(target_labels, kmeans_clusters.labels_)
        cs = completeness_score(target_labels, kmeans_clusters.labels_)
        vms = v_measure_score(target_labels, kmeans_clusters.labels_)
        aris = adjusted_rand_score(target_labels, kmeans_clusters.labels_)
        amis = adjusted_mutual_info_score(target_labels, kmeans_clusters.labels_)

        avg_svd_score = (hs + cs + vms + aris + amis) / 5

        print('Average Score: ' + str(avg_svd_score))

        if avg_svd_score > best_score_svd:
            best_score_svd = avg_svd_score
            best_r_svd = r

    return best_r_svd, best_score_svd

num_components = [1, 2, 3, 5, 10, 20, 50, 100, 300]
kmeans_cluster_model = KMeans(init='k-means++', max_iter=1000, n_clusters=20, n_i

best_r_svd, best_svd_score = calculate_best_svd_score(num_components, tfidf_dataf

print('Best r in terms of average score: ' + str(best_r_svd))
print('Best SVD Score: ' + str(best_svd_score))
```

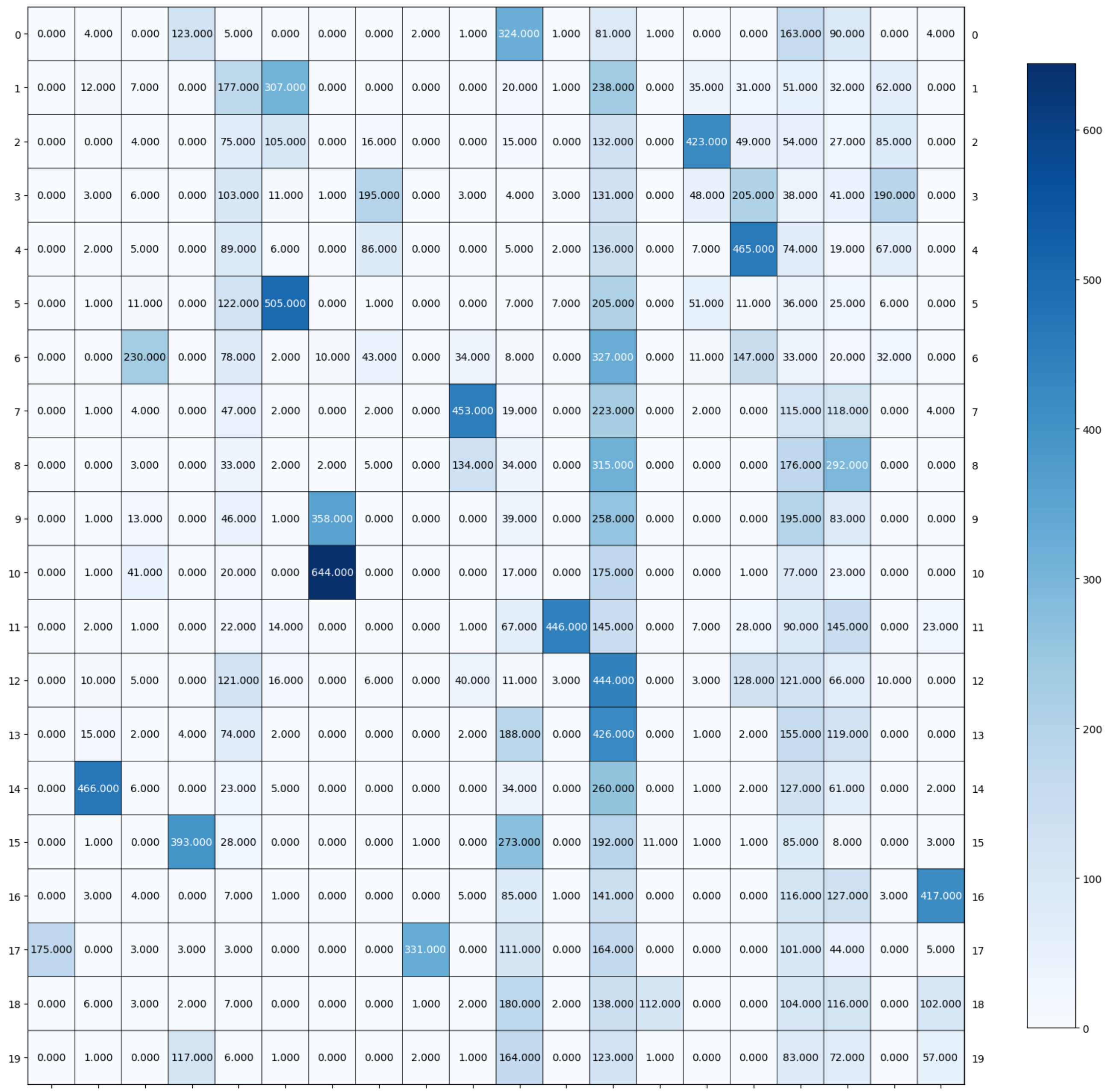
```
1
Average Score: 0.020699099772092347
2
Average Score: 0.1866356606182337
3
Average Score: 0.2210624664991326
5
Average Score: 0.2933199644577881
```

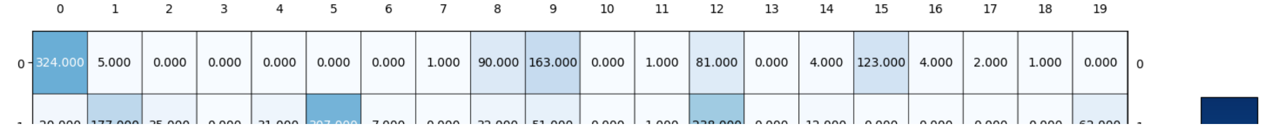
```
Average Score: 0.2935199644377881
10
Average Score: 0.29342498645198073
20
Average Score: 0.30886229818530025
50
Average Score: 0.30086532474510075
100
Average Score: 0.30106677274936217
300
Average Score: 0.2783574580258681
Best r in terms of average score: 20
Best SVD Score: 0.30886229818530025
```

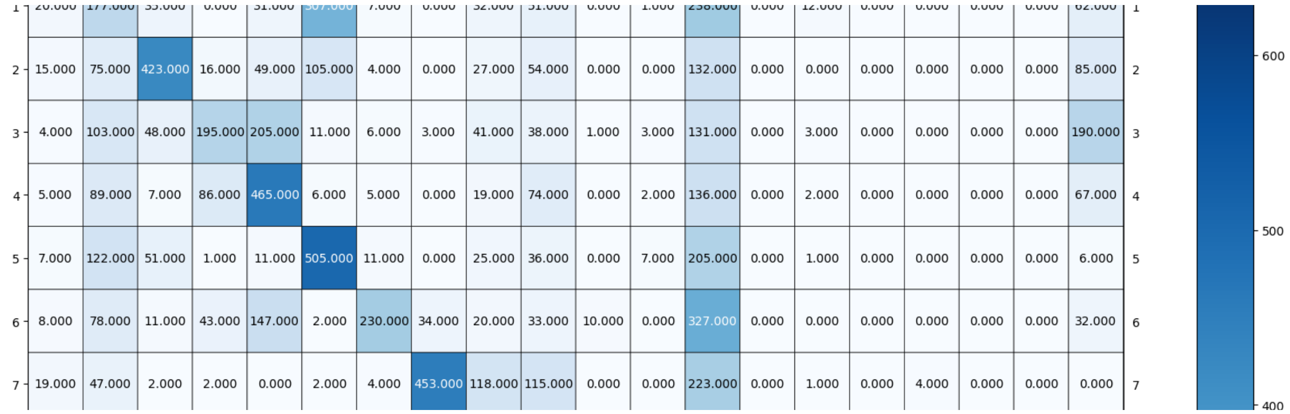
```
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.metrics import (
    homogeneity_score,
    completeness_score,
    v_measure_score,
    adjusted_rand_score,
    adjusted_mutual_info_score,
)
from sklearn.metrics import confusion_matrix
from scipy.optimize import linear_sum_assignment
from sklearn.metrics.cluster import contingency_matrix

def apply_svd_and_kmeans(data, num_components, kmeans_model):
    svd_transformer = TruncatedSVD(n_components=num_components, random_state=42)
    svd_features = svd_transformer.fit_transform(data)
    kmeans_model.fit(svd_features)

def evaluate_clustering_metrics(y_test, y_pred, name=""):
    print("Homogeneity score for %s: %f" % (name, homogeneity_score(y_test, y_pre
    print("Completeness score for %s: %f" % (name, completeness_score(y_test, y_p
    print("V-measure score for %s: %f" % (name, v_measure_score(y_test, y_pred)))
    print("Adjusted Rand Index score for %s: %f" % (name, adjusted_rand_score(y_t
    print("Adjusted mutual information score for %s: %f" % (name, adjusted_mutual


def visualize_confusion_matrix(target_labels, predicted_labels):
    cm = confusion_matrix(target_labels, predicted_labels)
    rows, cols = linear_sum_assignment(cm, maximize=True)
    plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, s

# R = 20 - Average Score
svd_r = 20
svd_model = TruncatedSVD(n_components=svd_r, random_state=42)
words_count_svd = svd_model.fit_transform(tfidf_dataframe)

kmeans_model = KMeans(init='k-means++', max_iter=1000, n_clusters=20, n_init=30,
apply_svd_and_kmeans(tfidf_dataframe, svd_r, kmeans_model)
```

```
apply_svd_and_kmeans(tfidf_dataframe, svd_r, kmeans_model)

# Evaluate metrics
evaluate_clustering_metrics(news_dataset.target, kmeans_model.labels_, name="SVD

# Visualize confusion matrix
plot_mat(contingency_matrix(news_dataset.target, kmeans_model.labels_), size = (1
visualize_confusion_matrix(news_dataset.target, kmeans_model.labels_)
```

```
Homogeneity score for SVD (r = 20): 0.336158
Completeness score for SVD (r = 20): 0.378021
V-measure score for SVD (r = 20): 0.355862
Adjusted Rand Index score for SVD (r = 20): 0.120619
Adjusted mutual information score for SVD (r = 20): 0.353652
```





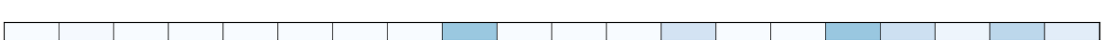


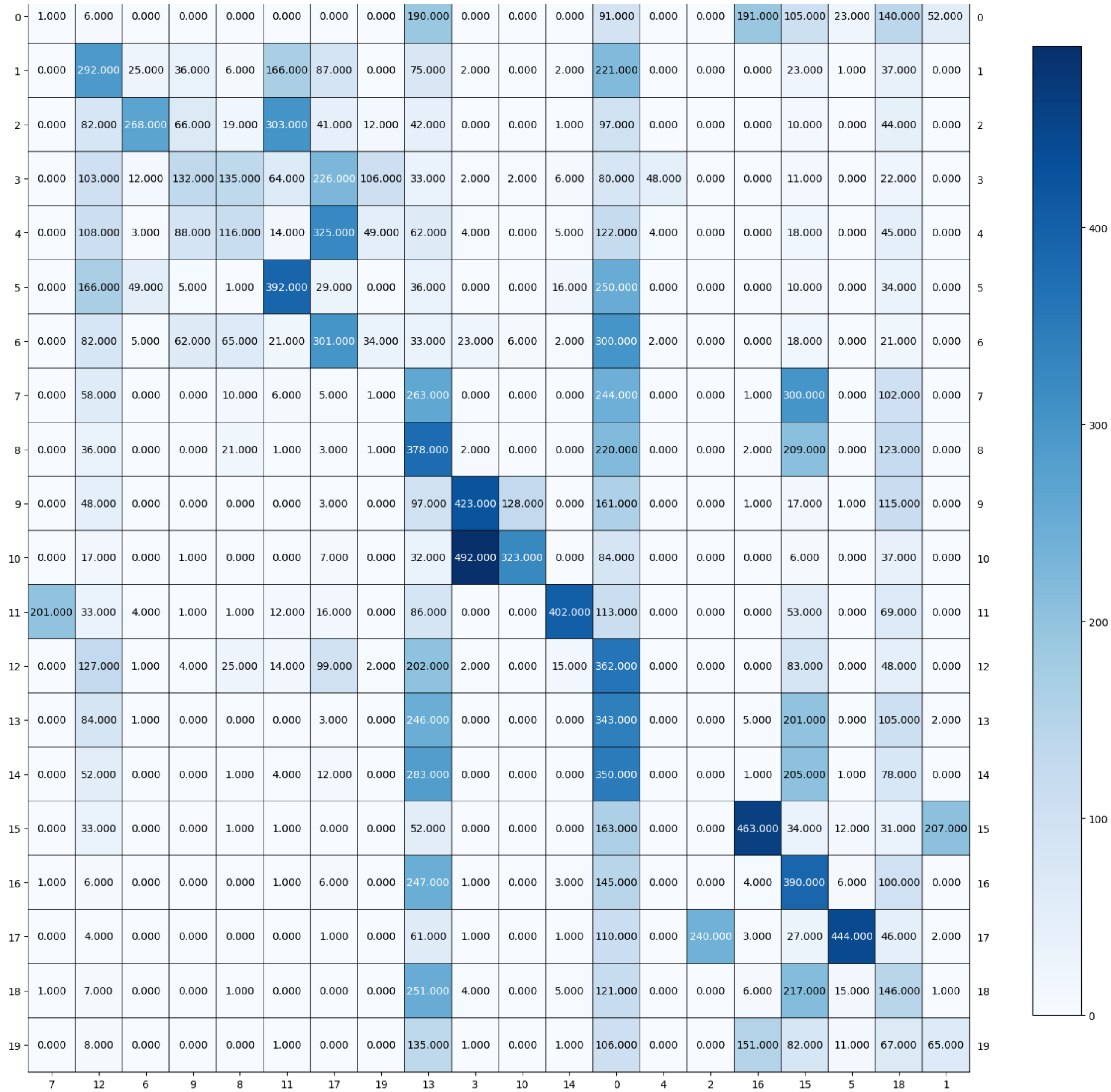

```
from sklearn.decomposition import NMF
from sklearn.cluster import KMeans
from sklearn.metrics import (
    homogeneity_score,
    completeness_score,
    v_measure_score,
    adjusted_rand_score,
    adjusted_mutual_info_score,
)
num_components = [1, 2, 3, 5, 10, 20, 50, 100, 300]
def calculate_best_nmf(components, data, target_labels, n_clusters=20, random_sta
    best_nmf_score = 0
    best_nmf_r = 0

    kmeans_model = KMeans(init='k-means++', max_iter=100000, n_clusters=n_cluster

    for r in components:
        nmf_model = NMF(n_components=r, init='random', random_state=random_state,
        words_count_nmf = nmf_model.fit_transform(data)
        kmeans_model.fit(words_count_nmf)

        hs = homogeneity_score(target_labels, kmeans_model.labels_)
        cs = completeness_score(target_labels, kmeans_model.labels_)
        vms = v_measure_score(target_labels, kmeans_model.labels_)
        aris = adjusted_rand_score(target_labels, kmeans_model.labels_)
        amis = adjusted_mutual_info_score(target_labels, kmeans_model.labels_)

        avg_score = (hs + cs + vms + aris + amis) / 5

        if avg_score > best_nmf_score:
            best_nmf_score = avg_score
            best_nmf_r = r

        print('Component ' + str(r) + ', ' + 'Average Score: ' + str(avg_score))

    return best_nmf_r, best_nmf_score

best_r_nmf, best_score_nmf = calculate_best_nmf(num_components, tfidf_dataframe,
```

```
best_r_nmf, best_score_nmf = calculate_best_nmf(num_components, tfidf_dataframe,

print('Best r for NMF: ' + str(best_r_nmf))
print('Best NMF Score: ' + str(best_score_nmf))
```

```
Component 1, Average Score: 0.02076162215956203
Component 2, Average Score: 0.1699054207666734
Component 3, Average Score: 0.20177601465736045
Component 5, Average Score: 0.2396382853881612
Component 10, Average Score: 0.2650333039858369
Component 20, Average Score: 0.2627086703407877
Component 50, Average Score: 0.23521298076051372
Component 100, Average Score: 0.14074921062990936
Component 300, Average Score: 0.05024577283103636
Best r for NMF: 10
Best NMF Score: 0.2650333039858369
```

```
from sklearn.metrics import confusion_matrix
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix
from scipy.optimize import linear_sum_assignment
from sklearn.decomposition import NMF
from sklearn.cluster import KMeans
from sklearn.metrics import (
    homogeneity_score,
    completeness_score,
    v_measure_score,
    adjusted_rand_score,
    adjusted_mutual_info_score,
)
def cluster_and_visualize(data, n_components, n_clusters=20, random_state=42):
    nmf_model = NMF(n_components=n_components, init='random', random_state=randon
    words_count_nmf = nmf_model.fit_transform(data)

    kmeans_model = KMeans(init='k-means++', max_iter=1000000, n_clusters=n_cluste
    kmeans_model.fit(words_count_nmf)

    confusion_mat = confusion_matrix(news_dataset.target, kmeans_model.labels_)
    rows, cols = linear_sum_assignment(confusion_mat, maximize=True)
    plot_mat(confusion_mat[rows[:, np.newaxis], cols], xticklabels=cols, yticklab

    print("Homogeneity score for %s: %f" % ("", homogeneity_score(news_dataset.ta
    print("Completeness score for %s: %f" % ("",  completeness_score(news_dataset
    print("V-measure score for %s: %f" % ("",  v_measure_score(news_dataset.targe
    print("Adjusted Rand Index score for %s: %f" % ("",  adjusted_rand_score(news
    print("Adjusted mutual information score for %s: %f" % ("",  adjusted_mutual_

# Usage
cluster_and_visualize(tfidf_dataframe, 10)
```






```
Homogeneity score for : 0.301093
Completeness score for : 0.346081
V-measure score for : 0.322023
Adjusted Rand Index score for : 0.101485
Adjusted mutual information score for : 0.319661
```

4. UMAP

Question 11

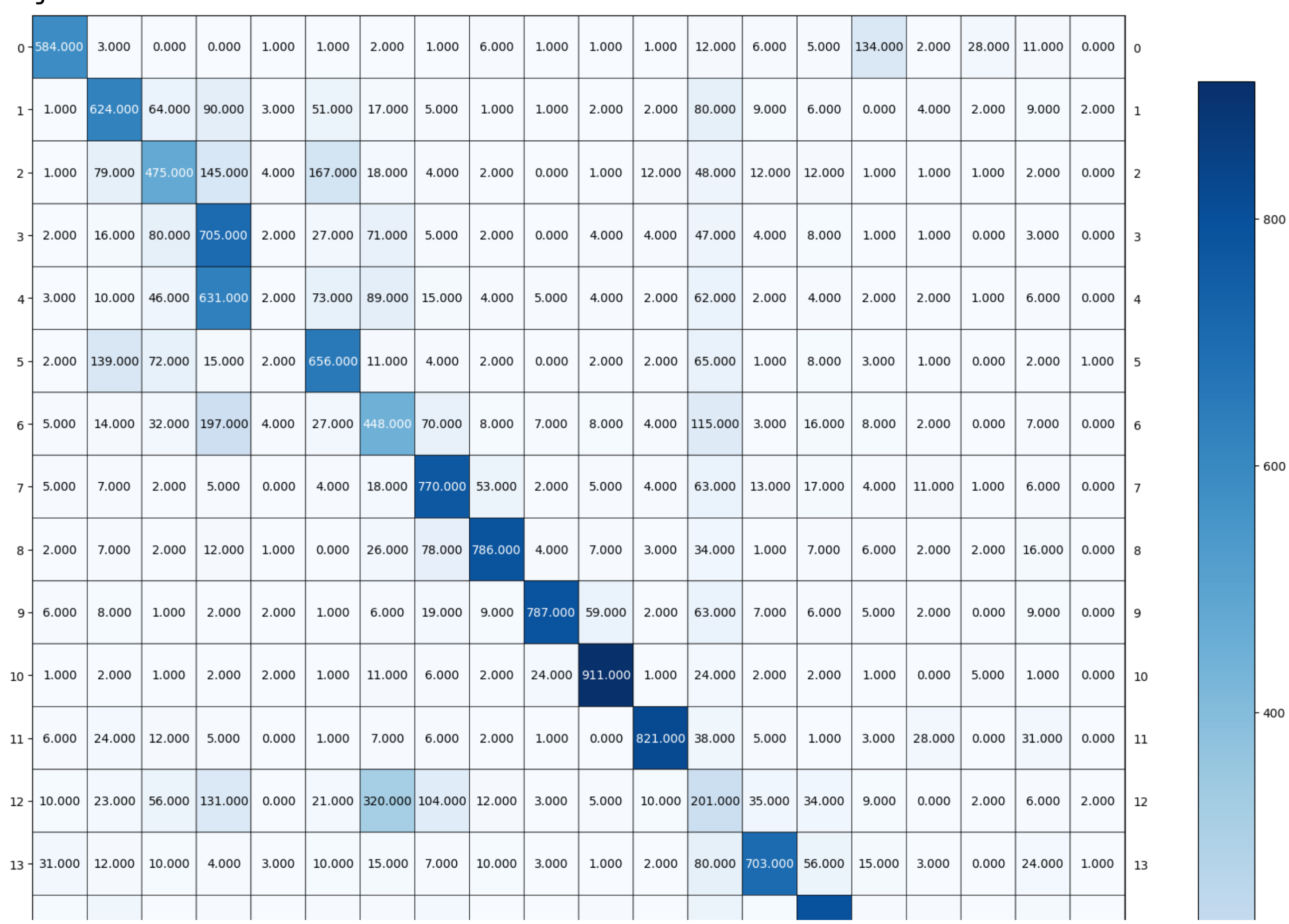
```
!pip uninstall umap
!pip install umap-learn
!pip install umap-learn[plot]
```

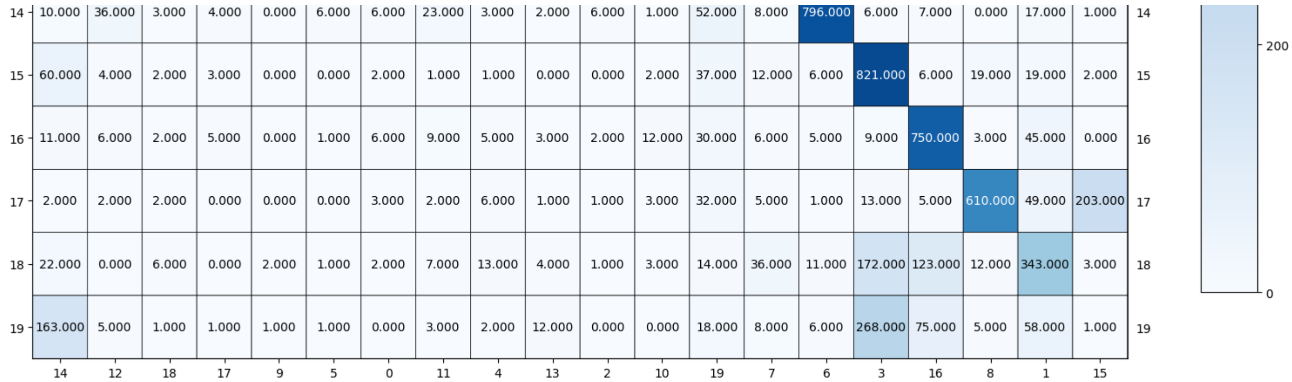
```
WARNING: Skipping umap as it is not installed.
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.9/90.9 kB 3.4 MB/s eta 0:00:
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.1
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dis
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.8/55.8 kB 5.7 MB/s eta 0:00:
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
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 si
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a3b431db
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
```

```
import umap.umap_ as umap
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from scipy.optimize import linear_sum_assignment

def run_umap_and_kmeans(tfidf_data, target_labels, distance_metric, n_components,
    print(f'\nUMAP Results using {distance_metric} & n_components = {n_components
    umap_model = umap.UMAP(n_components=n_components, metric=distance_metric, rar
    umap_transformed = umap_model.fit_transform(tfidf_data)

    kmeans_clusterer = KMeans(random_state=0, n_clusters=n_clusters, max_iter=10(
    kmeans_clusterer.fit(umap_transformed)

    print_cluster_metrics(target_labels, kmeans_clusterer.labels_)
    plot_confusion_matrix(target_labels, kmeans_clusterer.labels_)
```

```
def print_cluster_metrics(y_true, y_pred):
    print("Homogeneity score:", homogeneity_score(y_true, y_pred))
    print("Completeness score:", completeness_score(y_true, y_pred))
    print("V-measure score:", v_measure_score(y_true, y_pred))
    print("Adjusted Rand Index score:", adjusted_rand_score(y_true, y_pred))
    print("Adjusted Mutual Information score:", adjusted_mutual_info_score(y_true

def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    rows, cols = linear_sum_assignment(cm, maximize=True)
    plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, s

# Run UMAP for different parameters and metrics
umap_params_list = [(5, 'cosine'), (20, 'cosine'), (200, 'cosine'), (5, 'euclide
for n_components_value, distance_metric_value in umap_params_list:
    run_umap_and_kmeans(tfidf_dataframe, news_dataset.target, distance_metric_val
```

```
UMAP Results using cosine & n_components = 5:
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1943: UserWarning: n_jo
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state.
Homogeneity score: 0.5689113781606602
Completeness score: 0.58788157752841
V-measure score: 0.5782409320909574
Adjusted Rand Index score: 0.4524382375325882
Adjusted Mutual Information score: 0.5768448624199578
```






```
UMAP Results using cosine & n_components = 20:
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1943: UserWarning: n_jo
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state.
Homogeneity score: 0.5639415097627272
Completeness score: 0.5837578263288667
V-measure score: 0.5736785925777276
Adjusted Rand Index score: 0.44416336630818326
Adjusted Mutual Information score: 0.5722660045491073
```

Question 12

the first part highlights the superior performance of UMAP in dimensionality reduction, particularly when using the cosine metric. The robust diagonals in the contingency matrix and elevated metrics across various n_components emphasize its ability to achieve enhanced cluster separation compared to SVD/PCA and NMF. The selection of n_components=5, based on the highest V-score, is deemed optimal.

Conversely, the second part focuses on the suboptimal performance of UMAP dimensionality reduction with the Euclidean metric, indicated by low scores across homogeneity, completeness, v-measure, and adjusted random index for all n_component values. Despite its general unsuitability, the optimal setting is n_components=5, as observed in the confusion matrix, showing slightly improved cluster creation compared to other n_component values.

In conclusion, the cosine metric with n_components=5 remains the preferred choice for UMAP.

Question 13

```
from sklearn.cluster import KMeans
from sklearn.metrics import cluster
kmeans = KMeans(random_state=0, n_clusters=20, max_iter=1000, n_init=30)
kmeans.fit(tfidf_dataframe)
```

```
▾ KMeans
KMeans(max_iter=1000, n_clusters=20, n_init=30, random_state=0)
```

```
print("Homogeneity: ", cluster.homogeneity_score(news_dataset.target, kmeans.labe
print("Completeness: ",cluster. completeness_score(news_dataset.target, kmeans.la
print("V-measure: ", cluster.v_measure_score(news_dataset.target, kmeans.labels_)
print("Adjusted Rand-Index: ", cluster.adjusted_rand_score(news_dataset.target, k
print("Adjusted Mutual Information Score: ", cluster.adjusted_mutual_info_score(r
```

```
Homogeneity:  0.326807011612208
Completeness:  0.3743410597965642
V-measure:  0.3489627599775251
Adjusted Rand-Index:  0.11489276920106191
Adjusted Mutual Information Score:  0.3467089692894358
```

Clustering Algorithms that do not explicitly rely on the Gaussian distribution per cluster

Question 14-18

```
!pip install regex
!pip install nltk
!pip install sklearn
!pip uninstall umap
!pip install umap-learn
!pip install umap-learn[plot]
!pip install holoviews
!pip install -U ipykernel
!pip install hdbscan
```

```
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/d
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem
```

```
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
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:
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.1
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dis
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.8/55.8 kB 6.6 MB/s eta 0:00:
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
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 si
  Stored in directory: /root/.cache/pip/wheels/3a/70/07/428d2b58660a1a3b431db
Successfully built umap-learn
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.11 umap-learn-0.5.5
Requirement already satisfied: umap-learn[plot] in /usr/local/lib/python3.10/
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.1
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.10/
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa
```

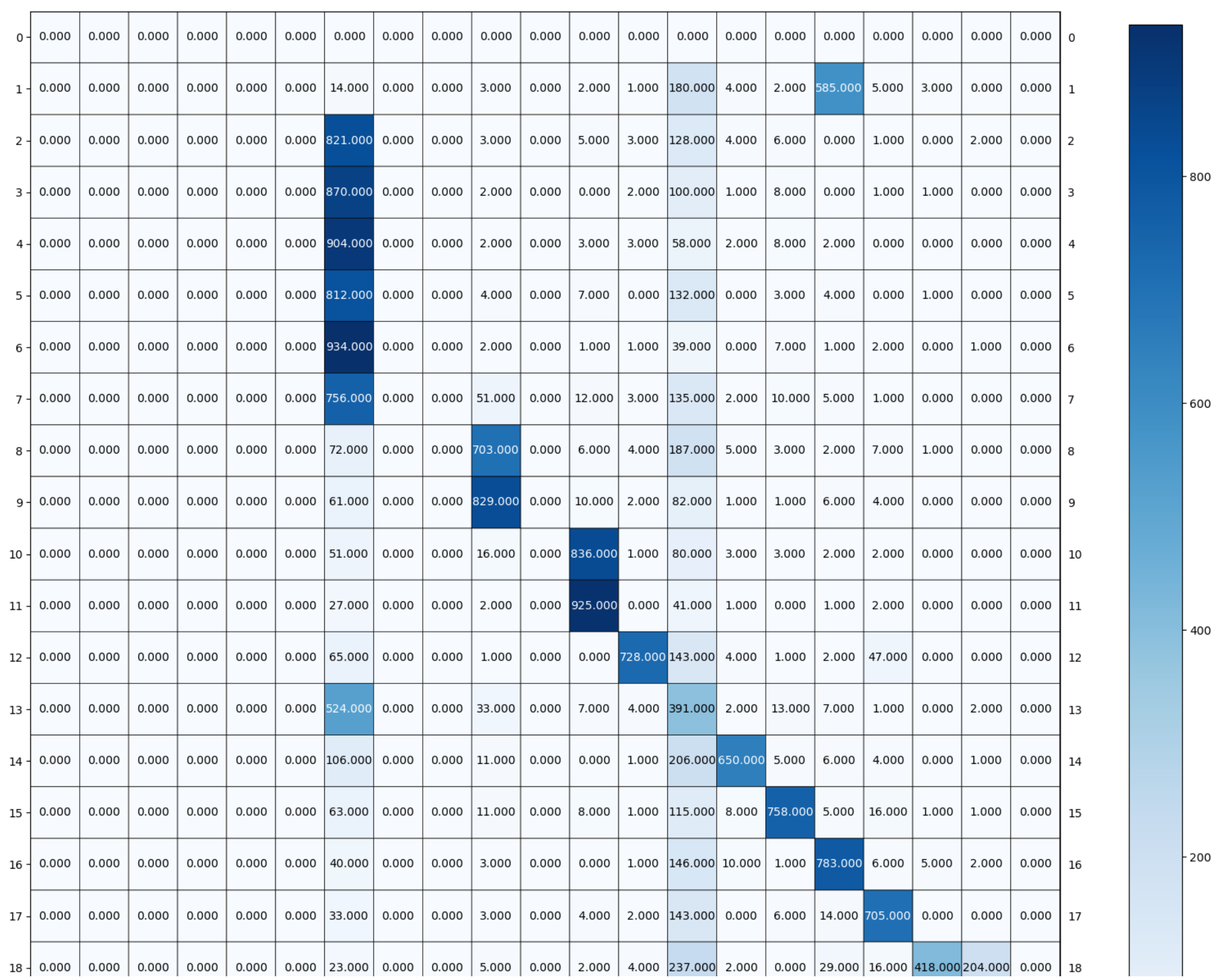
```
import numpy as np
import nltk, string
import itertools
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import sklearn
from sklearn.metrics import homogeneity_score, completeness_score, v_measure_scor
from sklearn.metrics import confusion_matrix
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.decomposition import TruncatedSVD, NMF
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
```

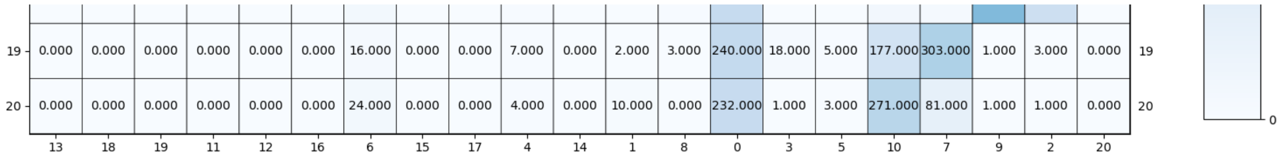
```
import pandas as pd
import umap.umap_ as umap
from scipy.optimize import linear_sum_assignment
import hdbscan


def plot_mat(mat, xticklabels = None, yticklabels = None, pic_fname = None, size=
             colorbar = True, grid = 'k', xlabel = None, ylabel = None, title = N
    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, edgecolc

    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)
```

```
        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_facec
            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, **k

    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()


def load_and_transform_dataset(min_df=3):

    # Create CountVectorizer and TfidfTransformer
    vectorizer = CountVectorizer(stop_words="english", min_df=min_df)
    transformer = TfidfTransformer(use_idf=True)

    # Transform the text data
    word_count_matrix = vectorizer.fit_transform(news_dataset.data)
    tfidf_matrix = transformer.fit_transform(word_count_matrix)
    tfidf_array = tfidf_matrix.toarray()

    # Get feature names from CountVectorizer
    feature_names = vectorizer.get_feature_names_out()

    # Create a DataFrame with the transformed data
    tfidf_dataframe = pd.DataFrame(data=tfidf_array, columns=feature_names)

    return tfidf_dataframe
```

```
def print_cluster_metrics(y_true, y_pred):
    print("Homogeneity score:", homogeneity_score(y_true, y_pred))
    print("Completeness score:", completeness_score(y_true, y_pred))
    print("V-measure score:", v_measure_score(y_true, y_pred))
    print("Adjusted Rand Index score:", adjusted_rand_score(y_true, y_pred))
    print("Adjusted Mutual Information score:", adjusted_mutual_info_score(y_true


def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    rows, cols = linear_sum_assignment(cm, maximize=True)
    plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, s


# Load the dataset
news_dataset = fetch_20newsgroups(subset = 'all',shuffle = True, random_state = 0

# Example usage
tfidf_dataframe = load_and_transform_dataset()
```

Question 14






```
# --------------------- Q14 -----------------------------------------------------
umap_model = umap.UMAP(n_components=20, metric="cosine", random_state=0)
umap_data = umap_model.fit_transform(tfidf_dataframe)

agg_ward_clusterer = AgglomerativeClustering(n_clusters=20, linkage="ward")
agg_ward_clusterer.fit_predict(umap_data)

print("Ward: ")
print_cluster_metrics(news_dataset.target, agg_ward_clusterer.labels_)

agg_single_clusterer = AgglomerativeClustering(n_clusters=20, linkage="single")
agg_single_clusterer.fit_predict(umap_data)

print("Single: ")
print_cluster_metrics(news_dataset.target, agg_single_clusterer.labels_)
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1943: UserWarning: n_jo
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state.
```

- The five clustering evaluation metrics for "ward" and "single" linkage criteria are shown below:

> | linkage | Homogeneity | Completeness | V-measure | Adjusted Rand Index | Adjusted Mutual Information |
> |---|---|---|---|---|---|
> | Ward | 0.556 | 0.587 | 0.571 | 0.422 | 0.569 |
> | Single | 0.018 | 0.363 | 0.034 | 0.001 | 0.029 |

- We can see that "ward" linkage performs much better than "single".

Question 15

```
# --------------------- Q15 -----------------------------------------------------
lst_components = [5, 20, 200]
lst_cluster_sizes = [20, 100, 200]
best_avg_score = 0
best_num_components = 0
best_min_cluster_size = 0
for num_components in lst_components:
    for min_cluster_size in lst_cluster_sizes:
        print("num_components: ", num_components, ", min_cluster_size: ", min_clu
        umap_model = umap.UMAP(n_components=num_components, metric="cosine", ranc
```

```
        umap_data = umap_model.fit_transform(tfidf_dataframe)
        hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size)
        hdbscan_clusterer.fit_predict(umap_data)
        print_cluster_metrics(news_dataset.target, hdbscan_clusterer.labels_)
        avg_score = (homogeneity_score(news_dataset.target, hdbscan_clusterer.lak
                     completeness_score(news_dataset.target, hdbscan_clusterer.lak
                     v_measure_score(news_dataset.target, hdbscan_clusterer.labels
                     adjusted_rand_score(news_dataset.target, hdbscan_clusterer.la
                     adjusted_mutual_info_score(news_dataset.target, hdbscan_clust
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_num_components = num_components
            best_min_cluster_size = min_cluster_size

print("best_num_components: ", best_num_components, " best_min_cluster_size: ", b
```

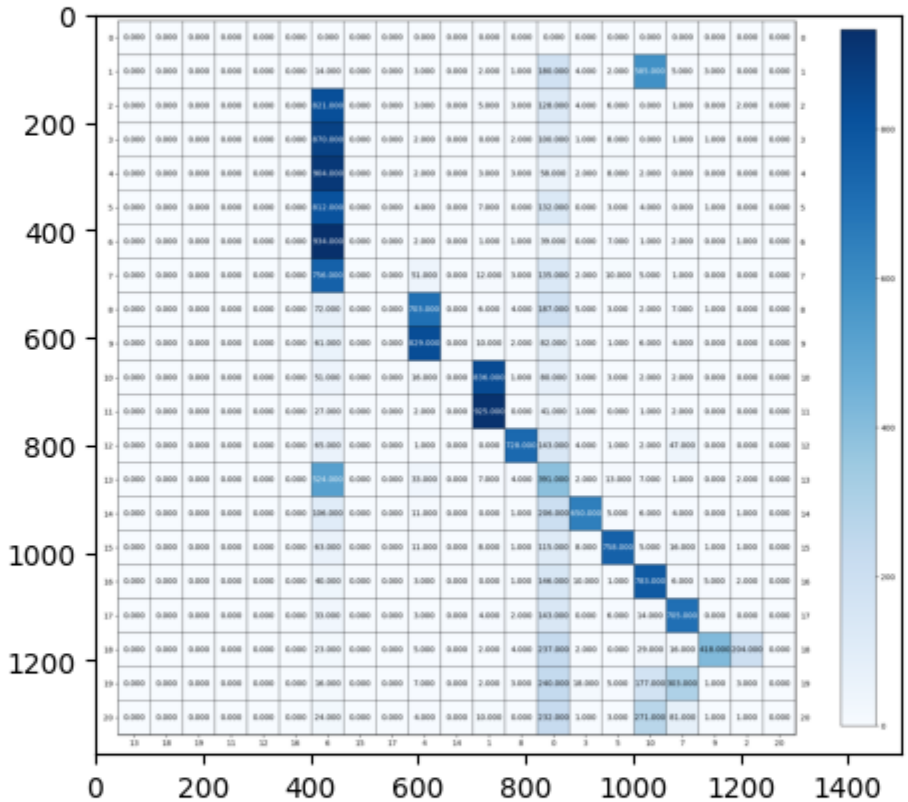
- The simulation resuls for different num_components and min_cluster_size are shown below.
- We found that best parameter combinations is num_components=5, min_cluster_size=100.

num_components	min_cluster_size	Homogeneity	Completeness	V-measure	Adjusted Rand Index	Adjusted Mu
5	20	0.429	0.443	0.436	0.077	0.423
5	100	0.429	0.629	0.510	0.227	0.509
5	200	0.419	0.614	0.498	0.219	0.497
20	20	0.437	0.453	0.445	0.086	0.433
20	100	0.418	0.627	0.502	0.232	0.501
20	200	0.415	0.608	0.493	0.213	0.492
200	20	0.424	0.445	0.435	0.075	0.422
200	100	0.421	0.614	0.499	0.209	0.498
200	200	0.412	0.611	0.490	0.208	0.489

Question 16

```
# --------------------- Q16 -----------------------------------------------------
umap_model = umap.UMAP(n_components=5, metric="cosine", random_state=0)
umap_data = umap_model.fit_transform(tfidf_dataframe)

hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=100)
hdbscan_clusterer.fit_predict(umap_data)

plot_confusion_matrix(news_dataset.target, hdbscan_clusterer.labels_)
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

images = mpimg.imread("saved_result.png")
plt.imshow(images)
plt.show()
```



- The confusion matrix is shown above, we can see that there are 21 clusters.
- "-1" means that the data point has not been classified to any clusters.

Question 17

```
# --------------------- Q17 -----------------------------------------------------
# --------------------- K-Means Only ---------------------
lst_k = [10, 20, 50]
best_avg_score = 0
best_k = 0
for k in lst_k:
    kmeans_clusterer = KMeans(n_clusters=k, max_iter=1000, n_init=30, random_stat
    kmeans_clusterer.fit(tfidf_dataframe)
    avg_score = (homogeneity_score(news_dataset.target, kmeans_clusterer.labels_)
                 completeness_score(news_dataset.target, kmeans_clusterer.labels_)
                 v_measure_score(news_dataset.target, kmeans_clusterer.labels_) +
                 adjusted_rand_score(news_dataset.target, kmeans_clusterer.labels_
                 adjusted_mutual_info_score(news_dataset.target, kmeans_clusterer.
```

```
        adjusted_mutual_info_score(news_dataset.target, kmeans_clusterer.
    if (avg_score > best_avg_score):
        best_avg_score = avg_score
        best_k = k
print("--------------------- K-Means Only ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_k: ", best_k)

# --------------------- Agglomerative Only ---------------------
agg_ward_clusterer = AgglomerativeClustering(n_clusters=20, linkage="ward")
agg_ward_clusterer.fit(tfidf_dataframe)
avg_score = (homogeneity_score(news_dataset.target, agg_ward_clusterer.labels_) +
            completeness_score(news_dataset.target, agg_ward_clusterer.labels_) +
            v_measure_score(news_dataset.target, agg_ward_clusterer.labels_) +
            adjusted_rand_score(news_dataset.target, agg_ward_clusterer.labels_)
            adjusted_mutual_info_score(news_dataset.target, agg_ward_clusterer.la
print("--------------------- Agglomerative Only ---------------------")
print("best_avg_score: ", avg_score)

# --------------------- HDBSCAN Only ---------------------
lst_k = [100, 200]
best_avg_score = 0
best_k = 0
for k in lst_k:
    hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
    hdbscan_clusterer.fit(tfidf_dataframe)
    avg_score = (homogeneity_score(news_dataset.target, hdbscan_clusterer.labels_
                completeness_score(news_dataset.target, hdbscan_clusterer.labels_
                v_measure_score(news_dataset.target, hdbscan_clusterer.labels_) +
                adjusted_rand_score(news_dataset.target, hdbscan_clusterer.labels
                adjusted_mutual_info_score(news_dataset.target, hdbscan_clusterer
    if (avg_score > best_avg_score):
        best_avg_score = avg_score
        best_k = k
print("--------------------- HDBSCAN Only ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_k: ", best_k)

# --------------------- SVD + K-Means ---------------------
lst_r = [5, 20, 200]
lst_k = [10, 20, 50]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    svd_model = TruncatedSVD(n_components=r, random_state=42)
    svd_data = svd_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        kmeans_clusterer = KMeans(n_clusters=k, max_iter=1000, n_init=30, random_
        kmeans_clusterer.fit(svd_data)
        avg_score = (homogeneity_score(news_dataset.target, kmeans_clusterer.labe
```

```
                    completeness_score(news_dataset.target, kmeans_clusterer.labe
                    v_measure_score(news_dataset.target, kmeans_clusterer.labels_
                    adjusted_rand_score(news_dataset.target, kmeans_clusterer.lab
                    adjusted_mutual_info_score(news_dataset.target, kmeans_cluste
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_r = r
            best_k = k
print("-------------------- SVD + K-Means --------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)
print("best_k: ", best_k)

# -------------------- SVD + Agglomerative --------------------
lst_r = [5, 20, 200]
best_avg_score = 0
best_r = 0
for r in lst_r:
    svd_model = TruncatedSVD(n_components=r, random_state=42)
    svd_data = svd_model.fit_transform(tfidf_dataframe)
    agg_ward_clusterer = AgglomerativeClustering(n_clusters=20, linkage="ward")
    agg_ward_clusterer.fit(svd_data)
    avg_score = (homogeneity_score(news_dataset.target, agg_ward_clusterer.labels
                completeness_score(news_dataset.target, agg_ward_clusterer.labels
                v_measure_score(news_dataset.target, agg_ward_clusterer.labels_)
                adjusted_rand_score(news_dataset.target, agg_ward_clusterer.label
                adjusted_mutual_info_score(news_dataset.target, agg_ward_clustere
    if (avg_score > best_avg_score):
        best_avg_score = avg_score
        best_r = r
print("-------------------- SVD + Agglomerative --------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)

# -------------------- SVD + HDBSCAN --------------------
lst_r = [5, 20, 200]
lst_k = [100, 200]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    svd_model = TruncatedSVD(n_components=r, random_state=42)
    svd_data = svd_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
        hdbscan_clusterer.fit(svd_data)
        avg_score = (homogeneity_score(news_dataset.target, hdbscan_clusterer.lab
                    completeness_score(news_dataset.target, hdbscan_clusterer.lab
                    v_measure_score(news_dataset.target, hdbscan_clusterer.labels
                    adjusted_rand_score(news_dataset.target, hdbscan_clusterer.la
```

```
                    adjusted_mutual_info_score(news_dataset.target, hdbscan_clust
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_r = r
            best_k = k
print("--------------------- SVD + HDBSCAN ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)
print("best_k: ", best_k)

# --------------------- NMF + K-Means ---------------------
lst_r = [5, 20, 200]
lst_k = [10, 20, 50]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    nmf_model = NMF(n_components=r, random_state=42, max_iter=1000)
    nmf_data = nmf_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        kmeans_clusterer = KMeans(n_clusters=k, max_iter=1000, n_init=30, random_
        kmeans_clusterer.fit(nmf_data)
        avg_score = (homogeneity_score(news_dataset.target, kmeans_clusterer.labe
                    completeness_score(news_dataset.target, kmeans_clusterer.labe
                    v_measure_score(news_dataset.target, kmeans_clusterer.labels_
                    adjusted_rand_score(news_dataset.target, kmeans_clusterer.lat
                    adjusted_mutual_info_score(news_dataset.target, kmeans_cluste
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_k = k
            best_r = r
print("--------------------- NMF + K-Means ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_k: ", best_k)
print("best_r: ", best_r)

# --------------------- NMF + Agglomerative ---------------------
lst_r = [5, 20, 200]
best_avg_score = 0
best_r = 0
for r in lst_r:
    nmf_model = NMF(n_components=r, random_state=42, max_iter=1000)
    nmf_data = nmf_model.fit_transform(tfidf_dataframe)
    agg_ward_clusterer = AgglomerativeClustering(n_clusters=20, linkage="ward")
    agg_ward_clusterer.fit(nmf_data)
    avg_score = (homogeneity_score(news_dataset.target, agg_ward_clusterer.labels
                completeness_score(news_dataset.target, agg_ward_clusterer.labels
                v_measure_score(news_dataset.target, agg_ward_clusterer.labels_)
                adjusted_rand_score(news_dataset.target, agg_ward_clusterer.label
                adjusted_mutual_info_score(news_dataset.target, agg_ward_clustere
```

```
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_r = r
print("--------------------- NMF + Agglomerative ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)

# --------------------- NMF + HDBSCAN ---------------------
lst_r = [5, 20, 200]
lst_k = [100, 200]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    nmf_model = NMF(n_components=r, random_state=42, max_iter=1000)
    nmf_data = nmf_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
        hdbscan_clusterer.fit(nmf_data)
        avg_score = (homogeneity_score(news_dataset.target, hdbscan_clusterer.lab
                     completeness_score(news_dataset.target, hdbscan_clusterer.lab
                     v_measure_score(news_dataset.target, hdbscan_clusterer.labels
                     adjusted_rand_score(news_dataset.target, hdbscan_clusterer.la
                     adjusted_mutual_info_score(news_dataset.target, hdbscan_clust
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_r = r
            best_k = k
print("--------------------- NMF + HDBSCAN ---------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)
print("best_k: ", best_k)

# --------------------- UMAP + K-Means ---------------------
lst_r = [5, 20, 200]
lst_k = [10, 20, 50]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    umap_model = umap.UMAP(n_components=r, metric="cosine", random_state=0)
    umap_data = umap_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        kmeans_clusterer = KMeans(n_clusters=k, max_iter=1000, n_init=30, random_
        kmeans_clusterer.fit(umap_data)
        avg_score = (homogeneity_score(news_dataset.target, kmeans_clusterer.labe
                     completeness_score(news_dataset.target, kmeans_clusterer.labe
                     v_measure_score(news_dataset.target, kmeans_clusterer.labels_
                     adjusted_rand_score(news_dataset.target, kmeans_clusterer.lab
                     adjusted_mutual_info_score(news_dataset.target, kmeans_cluste
        if (avg_score > best_avg_score):
```

```
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_k = k
            best_r = r
print("-------------------- UMAP + K-Means --------------------")
print("best_avg_score: ", best_avg_score)
print("best_k: ", best_k)
print("best_r: ", best_r)

# -------------------- UMAP + Agglomerative --------------------
lst_r = [5, 20, 200]
best_avg_score = 0
best_r = 0
for r in lst_r:
    umap_model = umap.UMAP(n_components=r, metric="cosine", random_state=0)
    umap_data = umap_model.fit_transform(tfidf_dataframe)
    agg_ward_clusterer = AgglomerativeClustering(n_clusters=20, linkage="ward")
    agg_ward_clusterer.fit(umap_data)
    avg_score = (homogeneity_score(news_dataset.target, agg_ward_clusterer.labels
                 completeness_score(news_dataset.target, agg_ward_clusterer.labels
                 v_measure_score(news_dataset.target, agg_ward_clusterer.labels_)
                 adjusted_rand_score(news_dataset.target, agg_ward_clusterer.label
                 adjusted_mutual_info_score(news_dataset.target, agg_ward_clustere
    if (avg_score > best_avg_score):
        best_avg_score = avg_score
        best_r = r
print("-------------------- UMAP + Agglomerative --------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)

# -------------------- UMAP + HDBSCAN --------------------
lst_r = [5, 20, 200]
lst_k = [100, 200]
best_avg_score = 0
best_r = 0
best_k = 0
for r in lst_r:
    umap_model = umap.UMAP(n_components=r, metric="cosine", random_state=0)
    umap_data = umap_model.fit_transform(tfidf_dataframe)
    for k in lst_k:
        hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=k)
        hdbscan_clusterer.fit(umap_data)
        avg_score = (homogeneity_score(news_dataset.target, hdbscan_clusterer.lab
                     completeness_score(news_dataset.target, hdbscan_clusterer.lab
                     v_measure_score(news_dataset.target, hdbscan_clusterer.labels
                     adjusted_rand_score(news_dataset.target, hdbscan_clusterer.la
                     adjusted_mutual_info_score(news_dataset.target, hdbscan_clust
        if (avg_score > best_avg_score):
            best_avg_score = avg_score
            best_r = r
            best_k = k
```

```
            best_k = k
print("-------------------- UMAP + HDBSCAN --------------------")
print("best_avg_score: ", best_avg_score)
print("best_r: ", best_r)
print("best_k: ", best_k)
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1943: UserWarning: n_jo
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state.
```

- The best combination is UMAP+K-means OR UMAP+Agglomerative, which is shown in the table below.

Dimensionality Reduction Method	Clustering Method	Best Average Score	Best r	Best k
None	K-Means	0.355	-	50
None	Agglomerative	0.343	-	-
None	HDBSCAN	0.2	-	100
SVD	K-Means	0.345	200	20
SVD	Agglomerative	0.335	20	-
SVD	HDBSCAN	0.2	5	100
NMF	K-Means	0.304	50	20
NMF	Agglomerative	0.320	20	-
NMF	HDBSCAN	0.2	5	200
UMAP	K-Means	0.559	20	200
UMAP	Agglomerative	0.559	200	-
UMAP	HDBSCAN	0.461	5	100

Part 2

Question 19-22

Question 19

Although trained on different dataset, feature leaned by VGG, especially the earlier layers, are able to capture generic visual pattern like edges and textures, which are applicable in other mage datasets. The features, can be generalized and still have discriminative power for clustering and classification on custom datasets.

Question 20 The helper code perform extraction by load pertained VGG 16 network. Then iterate through the image in flower dataset,. Each image is passed through VGG16 network to get feature representations. It extract features from one of last fully connected layers of VGG16. After passing image thought convolutional layers and average pooling layer. The extracted features then stored.

Question 21 Original image is 224 by 224 as indicated in the helper code. so total 50176 pixels per image. VGG16 extracted from one of its last fully connected layers, and result in feature vector with dimension of 4096 for each image, as indicated in starter.

Question 22 Extracted features from VGG are dense, which contains values for almost evert element. The features from TF-IDF are sparse features, and this means a lot of elements are 0. VGG features, each element usually contains non zero values, representing different learned visual patterns among images.

Question 13-25

Codes for q23:

```
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'):
        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=Tr
```

```
        self.features = list(vgg.features)
        self.features = nn.Sequential(*self.features)
        self.pooling = vgg.avgpool
        self.flatten = nn.Flatten()
        self.fc = vgg.classifier[0]

    def forward(self, x):
        out = self.features(x)
        out = self.pooling(out)
        out = self.flatten(out)
        out = self.fc(out)
        return out

assert torch.cuda.is_available()
feature_extractor = FeatureExtractor().cuda().eval()

dataset = datasets.ImageFolder(root='./flower_photos',
                               transform=transforms.Compose([transforms.Resiz
                                                             transforms.Cente
                                                             transforms.ToTer
                                                             transforms.Norma
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)

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)
```

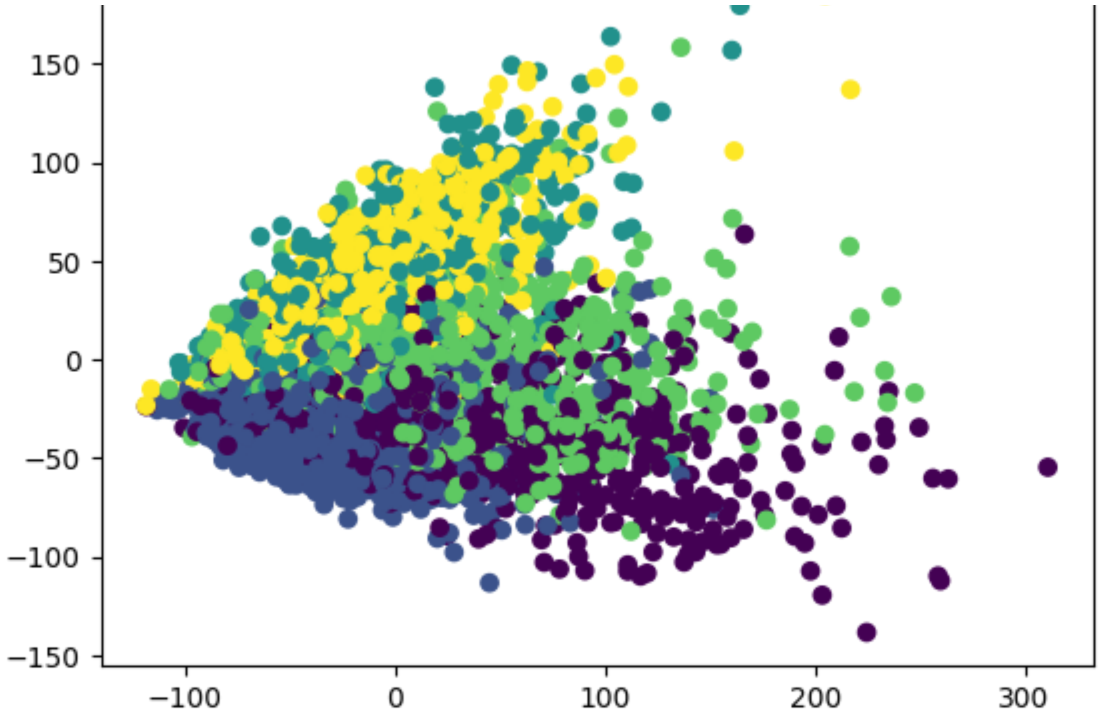
```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.ca
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: Use
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: Use
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /roo
100%|██████████| 528M/528M [00:04<00:00, 117MB/s]
100%|██████████| 58/58 [00:39<00:00,  1.46it/s]
```

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

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

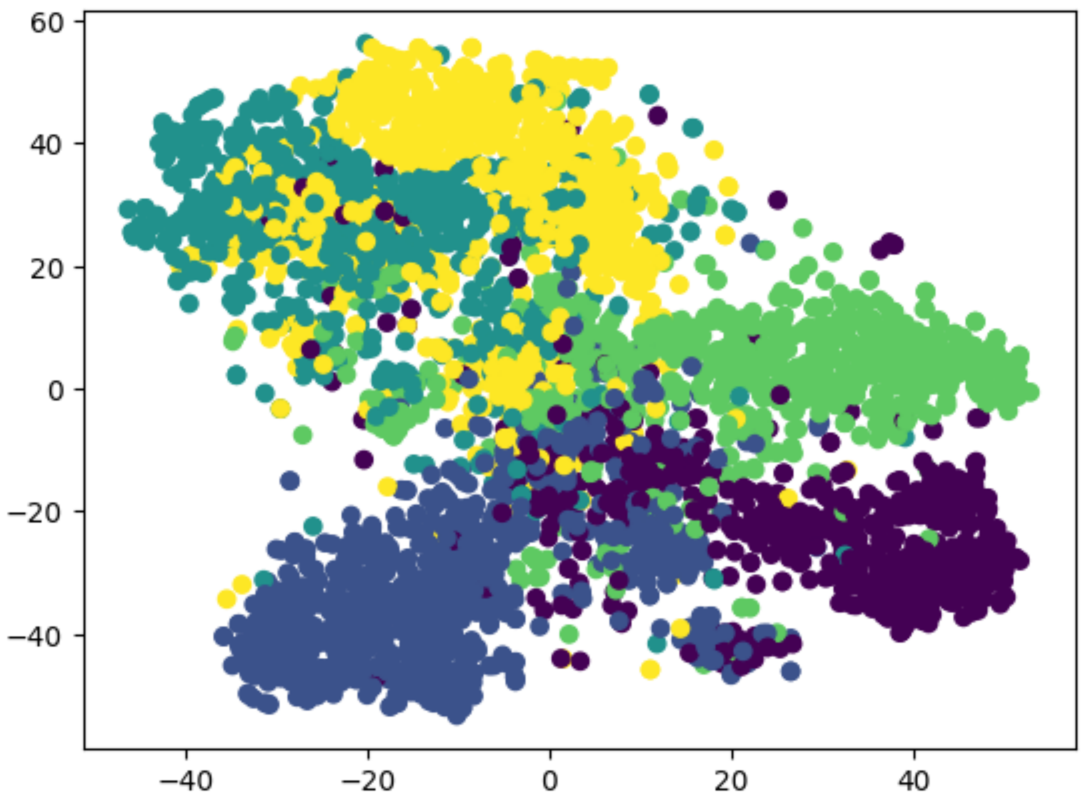
```
f_pca = PCA(n_components=2).fit_transform(f_all)
plt.scatter(*f_pca.T, c=y_all)
```

```
<matplotlib.collections.PathCollection at 0x7b01bc0ad420>
```




```
from sklearn.manifold import TSNE

f_tsne = TSNE(n_components=2).fit_transform(f_all)
plt.scatter(*f_tsne.T, c=y_all)
```

```
<matplotlib.collections.PathCollection at 0x7b01b0593520>
```



Q23 text answer:

Some classes are more seprated than others, for instance, purple ones are mostly isolated from other clusters.But green clusters overlaps with red clusters and blue clusters. This suggested that VGG network can distinguish some type of data better than others. Additionally, it can be noticed that some data points are outliers, and this means that they are awa from their own clusters because of some reaons, like noisy, blurry, or other issues with the photos.

More importantly, the second image shows a better clustering result than the first one, and this means that the second one have retained the information better.

Codes for q24:

```
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import requests
import os
import tarfile
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, adjusted_rand_score, adjusted_mutua
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin
from sklearn.cluster import AgglomerativeClustering
from hdbscan import HDBSCAN
import umap

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'):
        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('./')
```

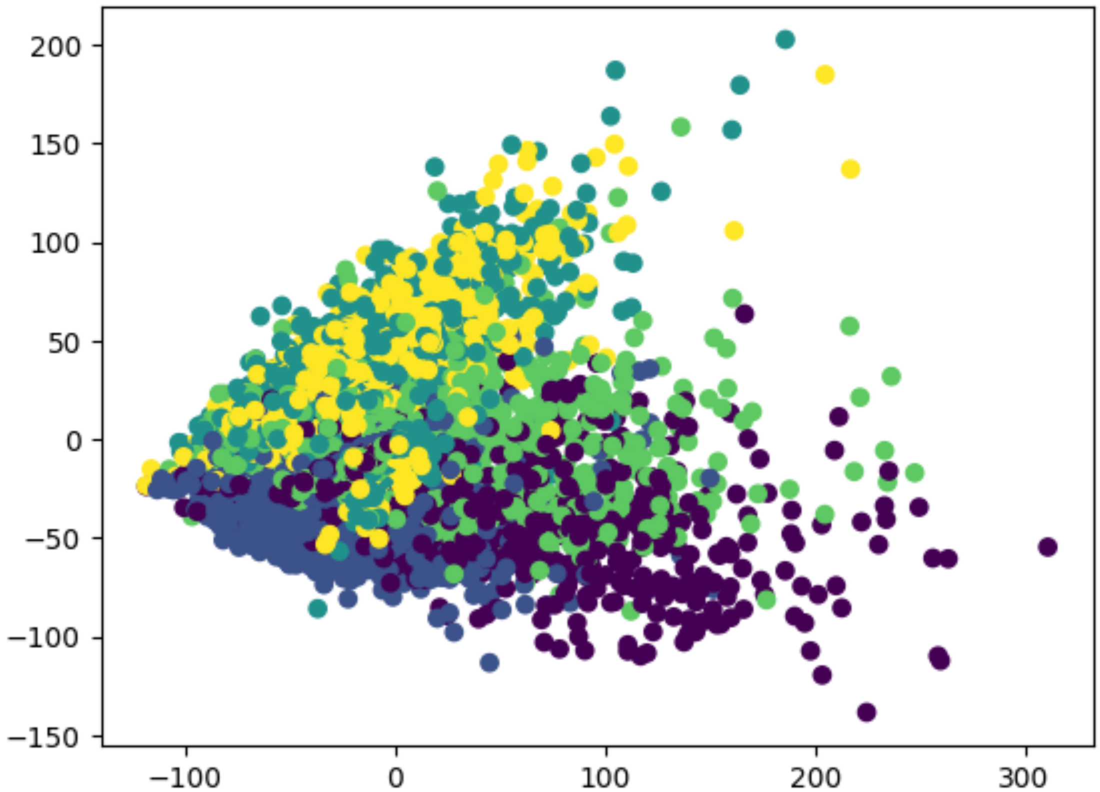
```
        os.remove('./flower_photos.tgz')

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

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

            self.features = list(vgg.features)
            self.features = nn.Sequential(*self.features)
            self.pooling = vgg.avgpool
            self.flatten = nn.Flatten()
            self.fc = vgg.classifier[0]

        def forward(self, x):
            out = self.features(x)
            out = self.pooling(out)
            out = self.flatten(out)
            out = self.fc(out)
            return out

    assert torch.cuda.is_available()
    feature_extractor = FeatureExtractor().cuda().eval()

    dataset = datasets.ImageFolder(root='./flower_photos',
                                   transform=transforms.Compose([transforms.Resiz
                                                                 transforms.Cente
                                                                 transforms.ToTer
                                                                 transforms.Norma

    dataloader = DataLoader(dataset, batch_size=64, shuffle=True)

    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)

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

f_pca = PCA(n_components=2).fit_transform(f_all)
plt.scatter(*f_pca.T, c=y_all)

# MLP Classifier
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-

        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()
        criterion = nn.NLLLoss()
        outputs = self.model(X_test)
        loss = criterion(outputs, y_test)
        return loss.item()

# Autoencoder
class Autoencoder(torch.nn.Module, TransformerMixin):
    def __init__(self, n_components):
        super().__init__()
        self.n_components = n_components
        self.n_features = None
        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, sel

    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-

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

        for epoch in tqdm(range(100)):
            for (X_,) in dataloader:
                X_ = X_.cuda()
                output = self(X_)
                loss = criterion(output, X_)
                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()

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

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.ca
100%|██████████| 58/58 [00:33<00:00,  1.74it/s]
(3670, 4096) (3670,)
100%|██████████| 100/100 [00:18<00:00,  5.28it/s]
<matplotlib.collections.PathCollection at 0x78cf902e1090>
```




```
labels_none = np.zeros_like(y_all)

# Clustering: SVD
svd = PCA(n_components=50)
X_svd = svd.fit_transform(f_all)
kmeans_svd = KMeans(n_clusters=5)
labels_svd = kmeans_svd.fit_predict(X_svd)

# Clustering: UMAP
umap_reducer = umap.UMAP(n_components=50)
X_umap = umap_reducer.fit_transform(f_all)
kmeans_umap = KMeans(n_clusters=5)
labels_umap = kmeans_umap.fit_predict(X_umap)
```

```
labels_umap = kmeans_umap.fit_predict(X_umap)

# Clustering: Autoencoder
autoencoder = Autoencoder(n_components=50).fit(f_all)
X_autoencoder = autoencoder.transform(f_all)
kmeans_autoencoder = KMeans(n_clusters=5)
labels_autoencoder = kmeans_autoencoder.fit_predict(X_autoencoder)

# Clustering: K-Means
kmeans_kmeans = KMeans(n_clusters=5)
labels_kmeans = kmeans_kmeans.fit_predict(f_all)

# Clustering: Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=5)
labels_agg = agg_clustering.fit_predict(f_all)

# Clustering: HDBSCAN
hdbscan = HDBSCAN(min_cluster_size=5, min_samples=5)
labels_hdbscan = hdbscan.fit_predict(f_all)

# Compute Rand scores
rand_scores = [
    adjusted_rand_score(y_all, labels_none),
    adjusted_rand_score(y_all, labels_svd),
    adjusted_rand_score(y_all, labels_umap),
    adjusted_rand_score(y_all, labels_autoencoder),
    adjusted_rand_score(y_all, labels_kmeans),
    adjusted_rand_score(y_all, labels_agg),
    adjusted_rand_score(y_all, labels_hdbscan)
]

print("Rand scores:")
print("None:", rand_scores[0])
print("SVD:", rand_scores[1])
print("UMAP:", rand_scores[2])
print("Autoencoder:", rand_scores[3])
print("K-Means:", rand_scores[4])
print("Agglomerative Clustering:", rand_scores[5])
print("HDBSCAN:", rand_scores[6])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
100%|██████████| 100/100 [00:19<00:00,  5.20it/s]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
  warnings.warn(
Rand scores:
None: 0.0
SVD: 0.19071616219335202
```

```
SVD: 0.19071616219335202
UMAP: 0.3975701600449636
Autoencoder: 0.2117826055669328
K-Means: 0.18919803381799868
Agglomerative Clustering: 0.2184499487113686
HDBSCAN: 0.006705947729476718
```

q24 text answer

Report the best result in terms of rand score within the table below:

None: 0.0

SVD: 0.19071616219335202

UMAP: 0.3975701600449636

Autoencoder: 0.2117826055669328

K-Means: 0.18919803381799868

Agglomerative Clustering: 0.2184499487113686

HDBSCAN: 0.006705947729476718

None: rand score is 0 , which means that no aggrement between the truth and the clustering label. This indicated that there is no clustering performed and all data point are single cluster.

SVD: the rand score indicates that some agreement between the truth and the clustering label. But it is not doing that well. UMAP: the rand score is relatively high between the truth and the labeled clustering. Compare to SVD, there is a improvement.

Auto-encoder: the rand score is between SVD and UMAP and there are moderate agreement between the truth and the labeling.

Means: the score is similar to SVD, and this means there is no significant improvement when comparing the SVD.

Agglomerative clustering:Slightly improvement compare to SVD but less the UMAP

HDBscan: the score is very low comparing to others, and this means HDBSCAN performs poorly when comparing the labeling and the truth.

Question 25

```
class MLP(torch.nn.Module):
    def __init__(self, num_features):
        super().__init__()
```

```
        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-

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

        for epoch in tqdm(range(100)):
            for (X_, y_) in dataloader:
                X_, y_ = X_.cuda(), y_.cuda()

                optimizer.zero_grad()
                outputs = self(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(X_test)
            _, predicted = torch.max(outputs, 1)
            accuracy = torch.sum(predicted == y_test).item() / len(y_test)

        return accuracy
```

```
class Autoencoder(torch.nn.Module, TransformerMixin):
    def __init__(self, n_components):
        super().__init__()
        self.n_components = n_components
        self.n_features = None
        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, sel

    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-

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

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

```
                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()
```

Double-click (or enter) to edit

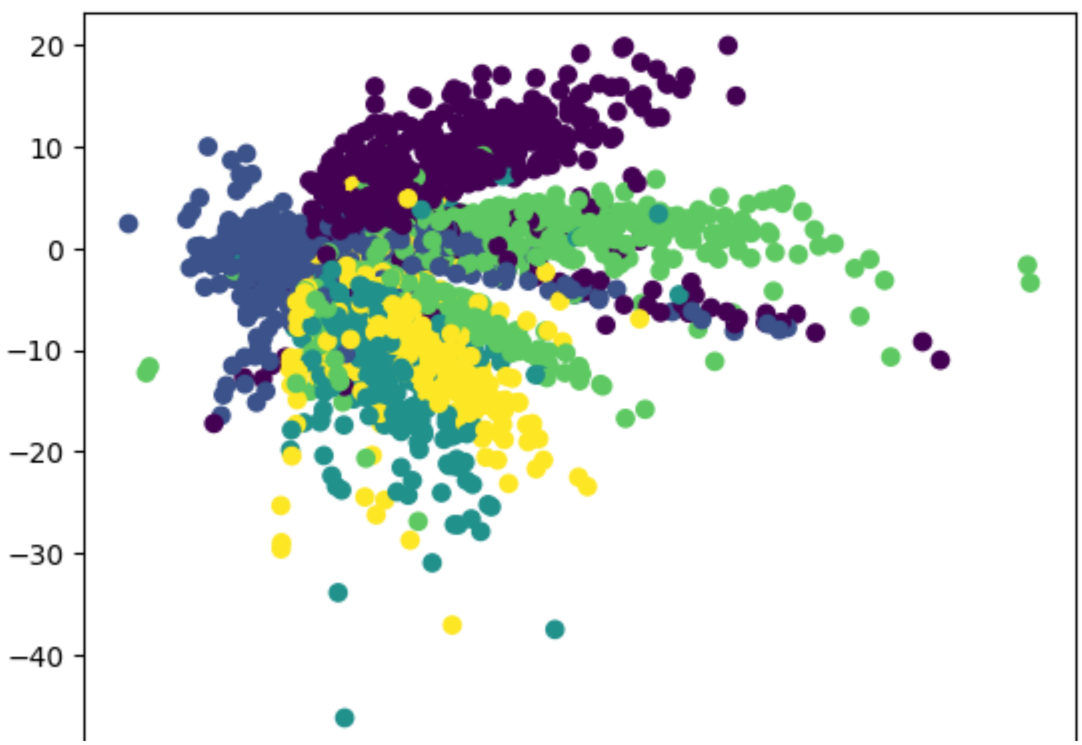
```
import os

filename = './flowers_features_and_labels.npz'

if os.path.exists(filename):
    file = np.load(filename)
    f_all, y_all = file['f_all'], file['y_all']

else:
    if not os.path.exists('./flower_photos'):
        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=Tr

            self.features = list(vgg.features)
            self.features = nn.Sequential(*self.features)
            self.pooling = vgg.avgpool
            self.flatten = nn.Flatten()
            self.fc = vgg.classifier[0]

        def forward(self, x):
            out = self.features(x)
            out = self.pooling(out)
            out = self.flatten(out)
            out = self.fc(out)
            return out

    assert torch.cuda.is_available()
```

```
feature_extractor = FeatureExtractor().cuda().eval()

dataset = datasets.ImageFolder(root='./flower_photos',
                               transform=transforms.Compose([transforms.Resiz
                                                             transforms.Cente
                                                             transforms.ToTer
                                                             transforms.Norma
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)

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])
```

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.ca
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: Use
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: Use
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /roo
100%|██████████| 528M/528M [00:04<00:00, 127MB/s]
100%|██████████| 58/58 [00:33<00:00,  1.71it/s]
```

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

```
100%|██████████| 100/100 [00:20<00:00,  4.81it/s]
<matplotlib.collections.PathCollection at 0x78d2e8d5ded0>
```



Question 25 text part:

The points in the graph are not seprate, and clusters are close together and cannot easily seprated. This means information is lost during the reduction process, so it suffers from dimentional reduction, and it is significant because that I see they got mixed together. This makes sense because clustering result in question 24 is not very good, and the reduced dimension feature fails to represent underlaying structure of the data, so it is expected that performance of MLP classifier suffer from reduced dimension feature.

Question 26-28

```
!unzip archive.zip
```

```
Archive:  archive.zip
  inflating: images/Abomasnow/0.jpg
  inflating: images/Abomasnow/1.jpg
  inflating: images/Abomasnow/2.jpg
  inflating: images/Abomasnow/3.jpg
  inflating: images/Abra/0.jpg
  inflating: images/Abra/1.jpg
  inflating: images/Abra/2.jpg
  inflating: images/Abra/3.jpg
  inflating: images/Absol/0.jpg
  inflating: images/Absol/1.jpg
  inflating: images/Absol/2.jpg
  inflating: images/Absol/3.jpg
  inflating: images/Accelgor/0.jpg
  inflating: images/Accelgor/1.jpg
  inflating: images/Aegislash/0.jpg
  inflating: images/Aegislash/1.jpg
  inflating: images/Aegislash/2.jpg
  inflating: images/Aegislash/3.jpg
  inflating: images/Aerodactyl/0.jpg
  inflating: images/Aerodactyl/1.jpg
  inflating: images/Aerodactyl/2.jpg
  inflating: images/Aerodactyl/3.jpg
  inflating: images/Aerodactyl/4.jpg
  inflating: images/Aerodactyl/5.jpg
  inflating: images/Aggron/0.jpg
  inflating: images/Aggron/1.jpg
  inflating: images/Aggron/2.jpg
  inflating: images/Aggron/3.jpg
  inflating: images/Aipom/0.jpg
  inflating: images/Aipom/1.jpg
  inflating: images/Aipom/2.jpg
```

```
  inflating: images/Alakazam/0.jpg
  inflating: images/Alakazam/1.jpg
  inflating: images/Alakazam/2.jpg
  inflating: images/Alakazam/3.jpg
  inflating: images/Alakazam/4.jpg
  inflating: images/Alakazam/5.jpg
  inflating: images/Alcremie/0.jpg
  inflating: images/Alcremie/1.jpg
  inflating: images/Alomomola/0.jpg
  inflating: images/Alomomola/1.jpg
  inflating: images/Altaria/0.jpg
  inflating: images/Altaria/1.jpg
  inflating: images/Altaria/2.jpg
  inflating: images/Altaria/3.jpg
  inflating: images/Amaura/0.jpg
  inflating: images/Amaura/1.jpg
  inflating: images/Ambipom/0.jpg
  inflating: images/Ambipom/1.jpg
  inflating: images/Amoonguss/0.jpg
  inflating: images/Amoonguss/1.jpg
  inflating: images/Ampharos/0.jpg
  inflating: images/Ampharos/1.jpg
  inflating: images/Ampharos/2.jpg
  inflating: images/Ampharos/3.jpg
  inflating: images/Ampharos/4.jpg
  inflating: images/Anorith/0.jpg
```

```
!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 6.3 MB/s eta 0:0
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
Collecting pyarrow>=12.0.0 (from datasets)
  Downloading pyarrow-15.0.0-cp310-cp310-manylinux_2_28_x86_64.whl (38.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.3/38.3 MB 14.6 MB/s eta 0:00
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/di
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 11.4 MB/s eta 0:
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packa
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 15.8 MB/s eta 0:
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/loc
```

```
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/pyth
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/pytho
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/pyth
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
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
ibis-framework 7.1.0 requires pyarrow<15,>=2, but you have pyarrow 15.0.0 whi
Successfully installed datasets-2.17.0 dill-0.3.8 multiprocess-0.70.16 pyarro
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packag
```

```
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
```

```
# load csv file and image paths to construct pokedex, use type_to_load=None to lc
def construct_pokedex(csv_path='Pokemon.csv', image_dir='./images/', type_to_loac
    pokedex = pd.read_csv(csv_path)
```

```
    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)

    # only keep pokemon with distinct id
    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=True)
    pokedex["Type2"] = pokedex["Type2"].str.strip()
    if type_to_load is not None:
        pokedex = pokedex[pokedex["Type1"].isin(type_to_load)].reset_index(drop=T
    return pokedex

# load clip model
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

# inference clip model on a list of image path
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).detach().cpu()
            image_embeddings += [image_embedding]

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

# inference clip model on a list of texts
def clip_inference_text(model, preprocess, texts, device):
    with torch.no_grad():
        text_embeddings = model.encode_text(clip.tokenize(texts).to(device)).deta
    text_embeddings /= np.linalg.norm(text_embeddings, axis=-1, keepdims=True)
    return text_embeddings
```

```
# compute similarity of texts to each image
def compute_similarity_text_to_image(image_embeddings, text_embeddings):
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.T), axis=-1)
    return similarity

# compute similarity of iamges to each text
def compute_similarity_image_to_text(image_embeddings, text_embeddings):
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.T), axis=0)
    return similarity

# Use TSNE to project CLIP embeddings to 2D space
def umap_projection(image_embeddings, n_neighbors=15, min_dist=0.1, metric='cosir
    distance_matrix = np.zeros((image_embeddings.shape[0], image_embeddings.shape
    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], image_embeddi
    distance_matrix = 1 - distance_matrix
    reducer = TSNE(n_components=2, metric="precomputed", init="random", random_st
    visualization_data = reducer.fit_transform(distance_matrix)
    return visualization_data

pokedex = construct_pokedex()
model, preprocess, device = load_clip_model()
image_embeddings = clip_inference_image(model, preprocess, pokedex["image_path"],
```

```
100%|██████████████████████████████████████| 890M/890M [00:11<00:00, 82.1MiB
100%|██████████| 753/753 [45:45<00:00,  3.65s/it]
```

Question 26

In this part, we created three types of text queries:

- "type: type_name" OR "type: type_name1 and type_name2"
- "type_name type Pokemon" OR "type_name1 and type_name2 type Pokemon"
- "Pokemon with type_name" OR "Pokemon with type_name1 and type_name2"

```
# "type: type_name"
# "type: type_name1 and type_name2"
def construct_query_typeA(texts1, texts2):
    query = []
    if texts2 == None:
        for i in range(len(texts1)):
            query.append("type: " + texts1[i])
    else:
        for i in range(len(texts1)):
```

```
            if texts2[i] == '':
                query.append("type: " + texts1[i])
            else:
                query.append("type: " + texts1[i] + " and " + texts2[i])
    query = pd.Series(query)
    return query

# "type_name type Pokemon"
# "type_name1 and type_name2 type Pokemon"
def construct_query_typeB(texts1, texts2):
    query = []
    if texts2 == None:
        for i in range(len(texts1)):
            query.append(texts1[i] + " type Pokemon")
    else:
        for i in range(len(texts1)):
            if texts2[i] == '':
                query.append(texts1[i] + " type Pokemon")
            else:
                query.append(texts1[i] + " and " + texts2[i] + " type Pokemon")
    query = pd.Series(query)
    return query

# "Pokemon with type_name"
# "Pokemon with type_name1 and type_name2"
def construct_query_typeC(texts1, texts2):
    query = []
    if texts2 == None:
        for i in range(len(texts1)):
            query.append("Pokemon with " + texts1[i])
    else:
        for i in range(len(texts1)):
            if texts2[i] == '':
                query.append("Pokemon with " + texts1[i])
            else:
                query.append("Pokemon with " + texts1[i] + " and " + texts2[i])
    query = pd.Series(query)
    return query

def avg_similarity(image_embeddings, text_embeddings):
    s1 = compute_similarity_image_to_text(image_embeddings, text_embeddings)
    s2 = compute_similarity_text_to_image(image_embeddings, text_embeddings)
    return (np.trace(s1)/s1.shape[0] + np.trace(s2)/s2.shape[0])/2

query_typeA = construct_query_typeA(pokedex["Type1"], pokedex["Type2"])
query_typeB = construct_query_typeB(pokedex["Type1"], pokedex["Type2"])
query_typeC = construct_query_typeC(pokedex["Type1"], pokedex["Type2"])
text_embeddings_A = clip_inference_text(model, preprocess, query_typeA, device)
text_embeddings_B = clip_inference_text(model, preprocess, query_typeB, device)
text_embeddings_C = clip_inference_text(model, preprocess, query_typeC, device)
```

```
print(avg_similarity(image_embeddings, text_embeddings_A))
print(avg_similarity(image_embeddings, text_embeddings_B))
print(avg_similarity(image_embeddings, text_embeddings_C))
```

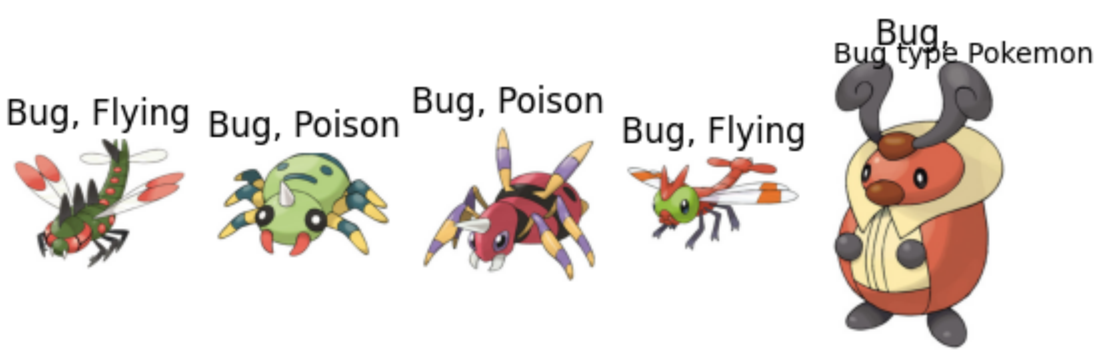
```
0.011364577775932402
0.011921476716222675
0.010087977050943362
```

- The results above show that the second type ("type_name type Pokemon" OR "type_name1 and type_name2 type Pokemon") is the most suitable template for queries.

```
import matplotlib.image as mpimg
```

```
def plot_images(type_name1, type_name2=None):
    query = construct_query_typeB(type_name1, type_name2)
    text_embeddings = clip_inference_text(model, preprocess, query, device)
    similarities = (text_embeddings @ image_embeddings.T)[0]
    index = np.argsort(similarities)[-5:][::-1].tolist()
    paths = pokedex["image_path"][index].tolist()
    images = [mpimg.imread(path) for path in paths]
    for i in range(len(images)):
        ax = plt.subplot(1, 5, i+1)
        plt.imshow(images[i])
        plt.axis('off')
        ax.set_title(pokedex["Type1"][index[i]] + ", " + pokedex["Type2"][index[i
    ax.text(0, 0, query[0])
    plt.show()
```

```
plot_images(["Bug"])
plot_images(["Grass"])
plot_images(["Fire"])
plot_images(["Dark"], ["Dragon"])
```



- The top five most relevant Pokemon for type "Bug", "Fire", "Grass" and "Dark and Dragon" are shown above.
- We found that the accuracies for "Bug", "Fire", "Grass" are much better than "Dark and Dragon", which is because there are some samples hard to classify because they look like dark dragon. While samples of "Bug", "Fire", "Grass" are more distinctive.

Question 27

```
import random


random.seed(42)
samples_index = random.sample(range(0, len(pokedex["image_path"])), 10)
samples_image_embeddings = clip_inference_image(model, preprocess, pokedex["image
```

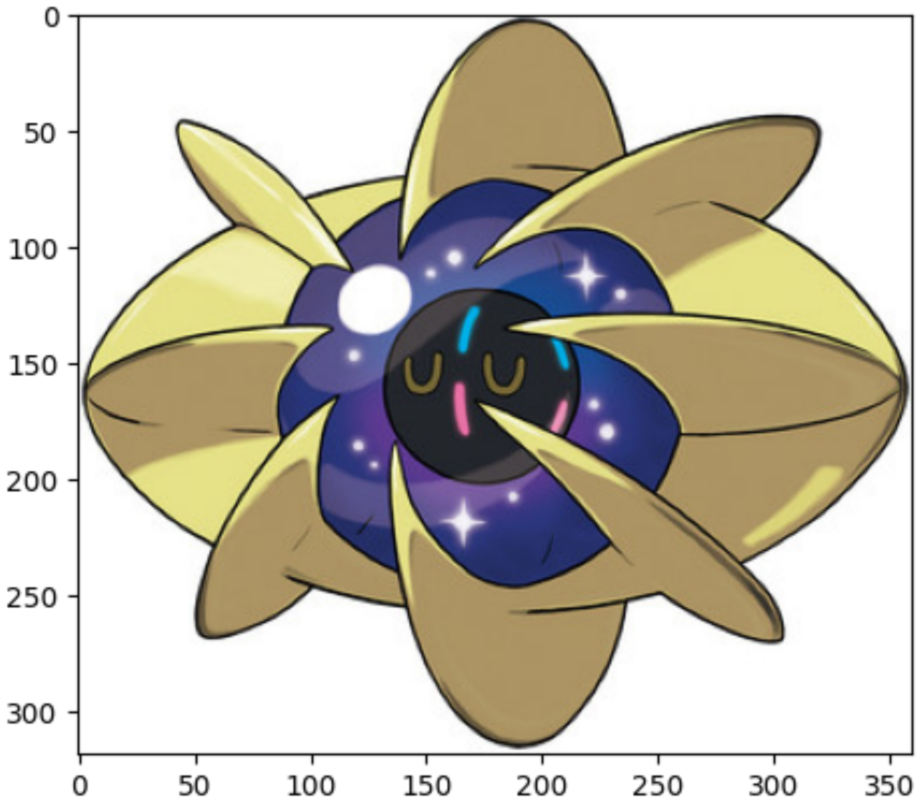
```
    100%|██████████| 10/10 [00:37<00:00,  3.71s/it]
```

```
similarities = (text_embeddings_B @ samples_image_embeddings.T).T
```

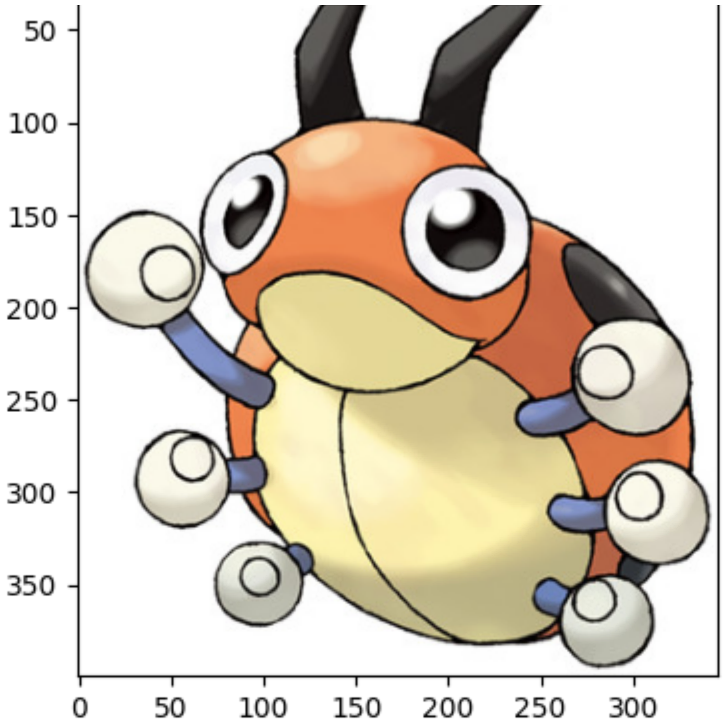
```
similarities.shape
```

```
(10, 753)
```

```
for i in range(similarities.shape[0]):
    index = np.argmax(similarities[i])
    path = pokedex["image_path"][samples_index[i]]
    plt.imshow(mpimg.imread(path))
    plt.show()
    print("Name: ", pokedex["Name"].tolist()[index])
    print("Type: ", pokedex["Type1"].tolist()[index], " ", pokedex["Type2"].tolis
    five_index = np.argsort(similarities)[i][-5:][::-1].tolist()
    print("Top five predicted types: ")
    for i in range(5):
        print(pokedex["Type1"][five_index[i]], )
    print()
```



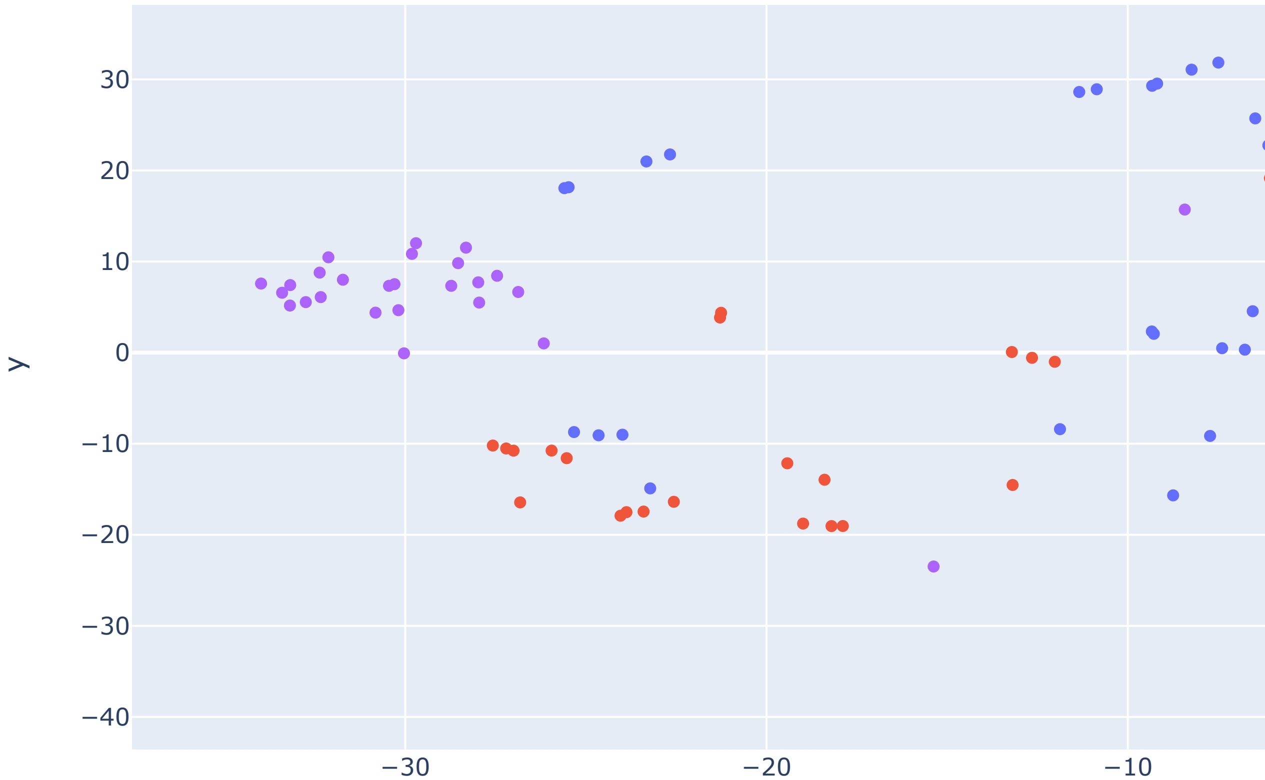

```
Name:  Inkay
Type:  Dark   Psychic
Top five predicted types:
Dark
Dark
Dark
Dark
Dark
```



- 10 samples of Pokemon images, name, type and predicted types are shown above.

Question 28

```
import plotly.express as px


umap_data = umap_projection(image_embeddings)

pokedex["x"] = umap_data[:,0]
pokedex["y"] = umap_data[:,1]


lst_color = []
for i in range(len(pokedex)):
    if pokedex["Type1"][i] == "Bug" or pokedex["Type1"][i] == "Fire" or pokedex["
        lst_color.append(pokedex["Type1"].iloc[i])
    else:
        lst_color.append(None)
pokedex["color"] = pd.Series(lst_color)

fig = px.scatter(pokedex, x="x", y="y", color="color", hover_data=["Name", "Type1
fig.show()
```



- The clusters for Bug, Fire, and Grass are shown above, where the x and y coordinates represent the vectors of the projected data.
- We can see that some clusters are formed for Bug (purple) and Grass (Blue), but there are many data points that are not clustered evidently.