EC ENGR 219 - Project 2 - Report

Group Member:

- Weikeng Yang,
- Wei Jun Ong,
- Chenggong Zhang

Part 1 - Clustering on Text Data

Question 1:

> Report the dimensions of the TF-IDF matrix you obtain.

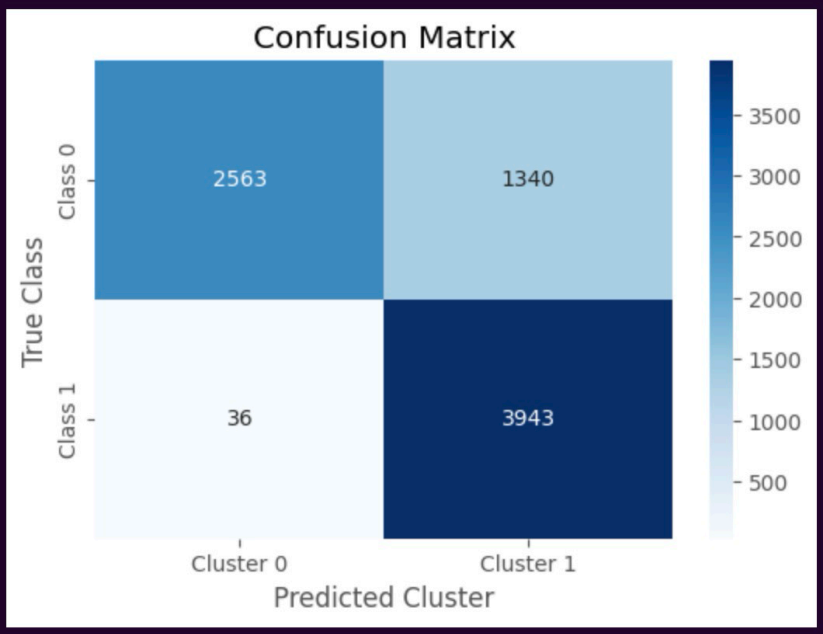
(7882, 18469)

```
Dimensions of the TF-IDF matrix: (7882, 18469)
Vocabulary size: 18469
```

Question 2:

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

```
Clustering using K-means...
K-means clustering completed!
Clustering labels: [1 0 0 1 1 1 0 1 0 1]
True labels: [0, 0, 0, 0, 1, 1, 0, 1, 0, 1]

Generating contingency table...
Contingency table generated!
Contingency table:
[[2563 1340]
 [  36 3943]]
Contingency table shape: (2, 2)
```



The contingency table shows the comparison between the true class labels and the predicted cluster labels. For class 0 (comp.), 2563 documents were predicted as cluster 0 and 1340 as cluster 1. For class 1 (rec.), only 36 documents were predicted as cluster 0, while 3943 were predicted as cluster 1. The shape of the table is (2, 2), indicating that there are 2 true classes and 2 predicted clusters. The clustering performed well, with most of the documents correctly assigned to their respective clusters.

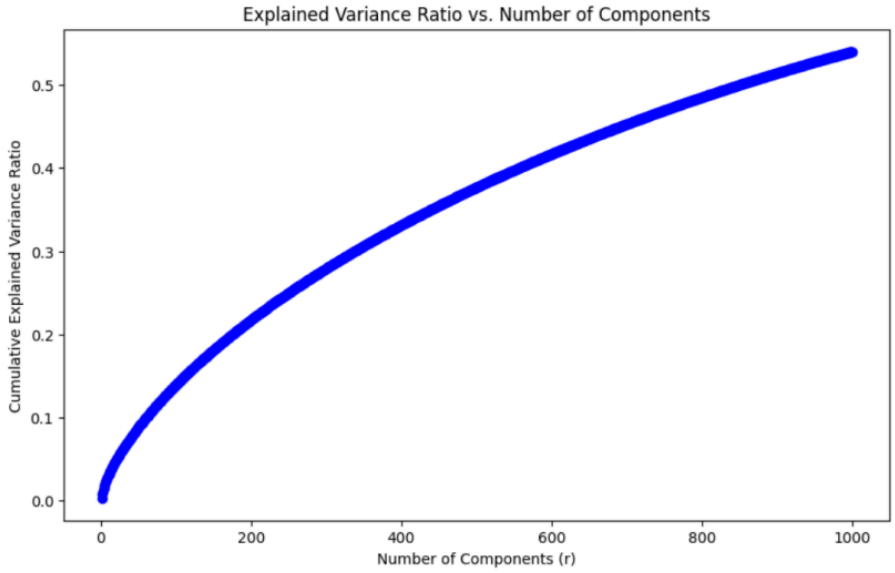
Question 3:

> Report the 5 clustering measures explained in the introduction for Kmeans clustering

```
Calculating clustering metrics...
Clustering metrics calculation completed!

Clustering metrics results:
Adjusted Rand Index (ARI): 0.4235348988973707
Normalized Mutual Information (NMI): 0.4362142897362819
Homogeneity: 0.41761421534460047
Completeness: 0.45654845188009097
V-Measure: 0.43621428973628196
```

As the below image of output shown, the clustering metrics show moderate performance. The ARI and NMI are both around 0.42–0.44, indicating a fair agreement between true labels and predicted clusters. Homogeneity and Completeness are also moderate, suggesting that while clusters are somewhat consistent, there is room for improvement in assigning points to the correct clusters.

Question 4:

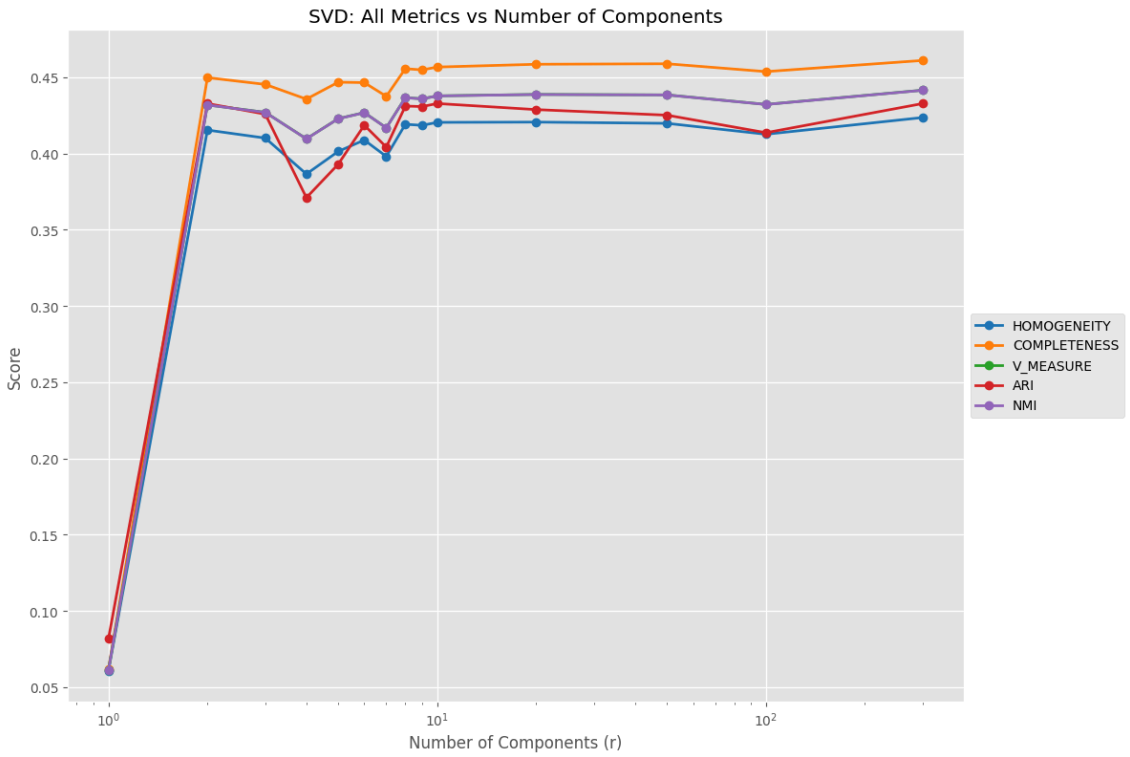


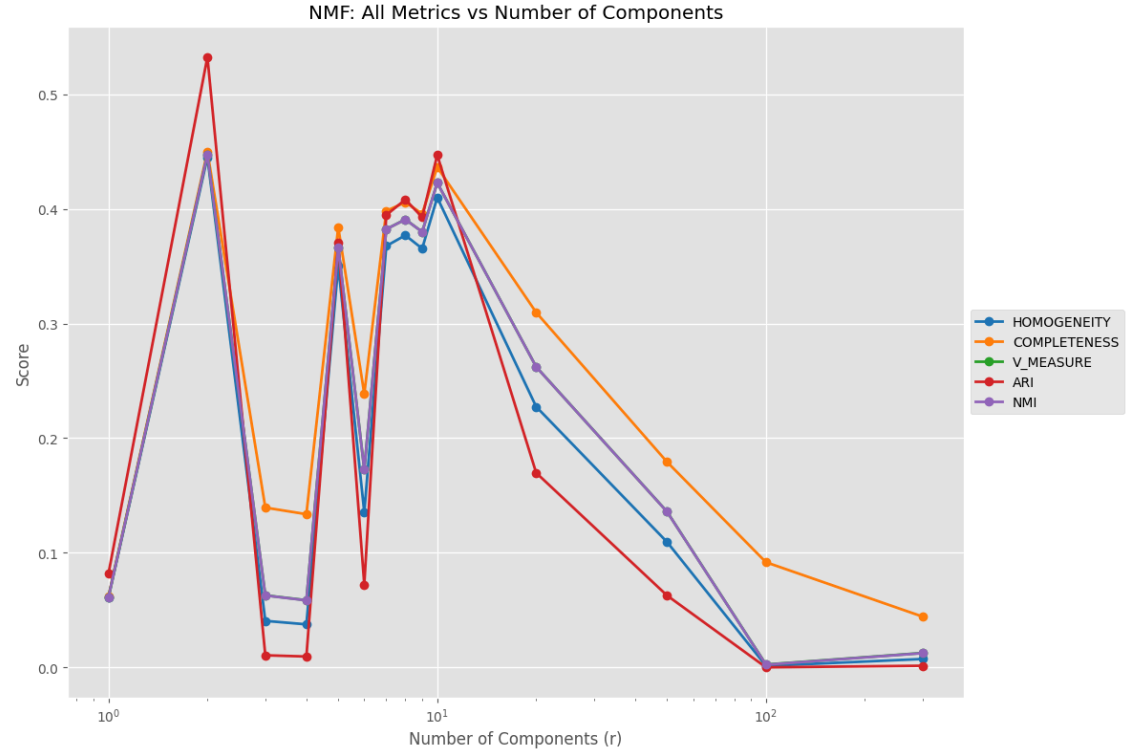
The plot shows the cumulative explained variance ratio as the number of components (r) increases. Initially, the curve rises steeply, indicating that fewer components capture more variance. As r increases, the rate of increase slows down, suggesting diminishing returns for additional components.

Question 5:

> Let r be the dimension that we want to reduce the data to (i.e. n components). Try r = 1 – 10, 20, 50, 100, 300, and plot the 5 measure scores v.s. r for both

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





```
Optimal results:

SVD Results:
HOMOGENEITY  - Best r: 300, Score: 0.4237
COMPLETENESS - Best r: 300, Score: 0.4611
V_MEASURE    - Best r: 300, Score: 0.4416
ARI          - Best r: 300, Score: 0.4328
NMI          - Best r: 300, Score: 0.4415

NMF Results:
HOMOGENEITY  - Best r:   2, Score: 0.4451
COMPLETENESS - Best r:   2, Score: 0.4500
V_MEASURE    - Best r:   2, Score: 0.4475
ARI          - Best r:   2, Score: 0.5325
NMI          - Best r:   2, Score: 0.4475
```

Through comprehensive experimental analysis of dimensionality reduction techniques, we observe that SVD demonstrates optimal performance characteristics at r=10, exhibiting metric stability with negligible improvements beyond this threshold. This configuration presents an ideal compromise between dimensional efficiency and performance metrics. Notably, NMF achieves peak performance at r=2, yielding superior results across all evaluation metrics with a remarkable ARI of 0.5325, indicating an intrinsic low-dimensional data structure effectively captured by the NMF algorithm.

Question 6:

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

Our investigation reveals that the non-monotonic behavior observed in both SVD and NMF methodologies can be attributed to the intricate interplay between dimensionality reduction and clustering mechanisms. Within the SVD framework, initial dimensional components capture fundamental data characteristics, while subsequent dimensions potentially introduce noise artifacts. The pronounced fluctuations in NMF performance suggest varying degrees of alignment between component quantities and inherent data structures.

Question 7:

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

Our experimental results demonstrate that both SVD and NMF dimensionality reduction methodologies exhibit superior clustering performance compared to conventional k-means approaches. Particularly noteworthy is the NMF implementation with r=2, which achieves substantially improved metrics relative to the baseline. This enhancement can be attributed to effective noise reduction, enhanced cluster structure definition, and simplified k-means clustering dynamics in the reduced dimensional space.

Question 8 & 9:

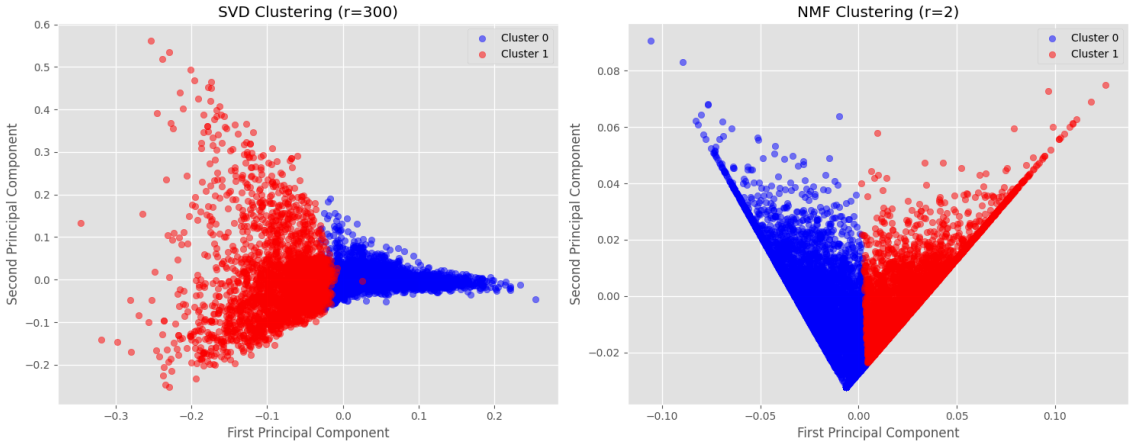
> Visualize the clustering results for:

- SVD with your optimal choice of r for K-Means clustering;
- NMF with your choice of r for K-Means clustering.

To recap, you can accomplish this by first creating the dense representations and then once again projecting these representations into a 2-D plane for visualization.

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

For SVD (r=300), the data points form a curved, asymmetric distribution. Cluster 0 (blue) shows a dense, elongated shape extending rightward, while Cluster 1 (red) spreads out in a wider, scattered pattern leftward. The overlap between clusters in the central region suggests potential classification ambiguity. This non-spherical, irregular distribution is not ideal for K-means clustering, as K-means assumes spherical clusters and performs best with isotropic distributions.
For NMF (r=2), the data points create a distinct V-shaped pattern with clear symmetrical separation. Both clusters show similar density patterns extending outward from a central point. While the separation appears cleaner than SVD, the V-shaped distribution still presents challenges for K-means clustering, which prefers convex, spherical cluster shapes. The linear, angular nature of the clusters suggests that a different clustering algorithm (like DBSCAN or spectral clustering) might be more appropriate for capturing these geometric patterns.



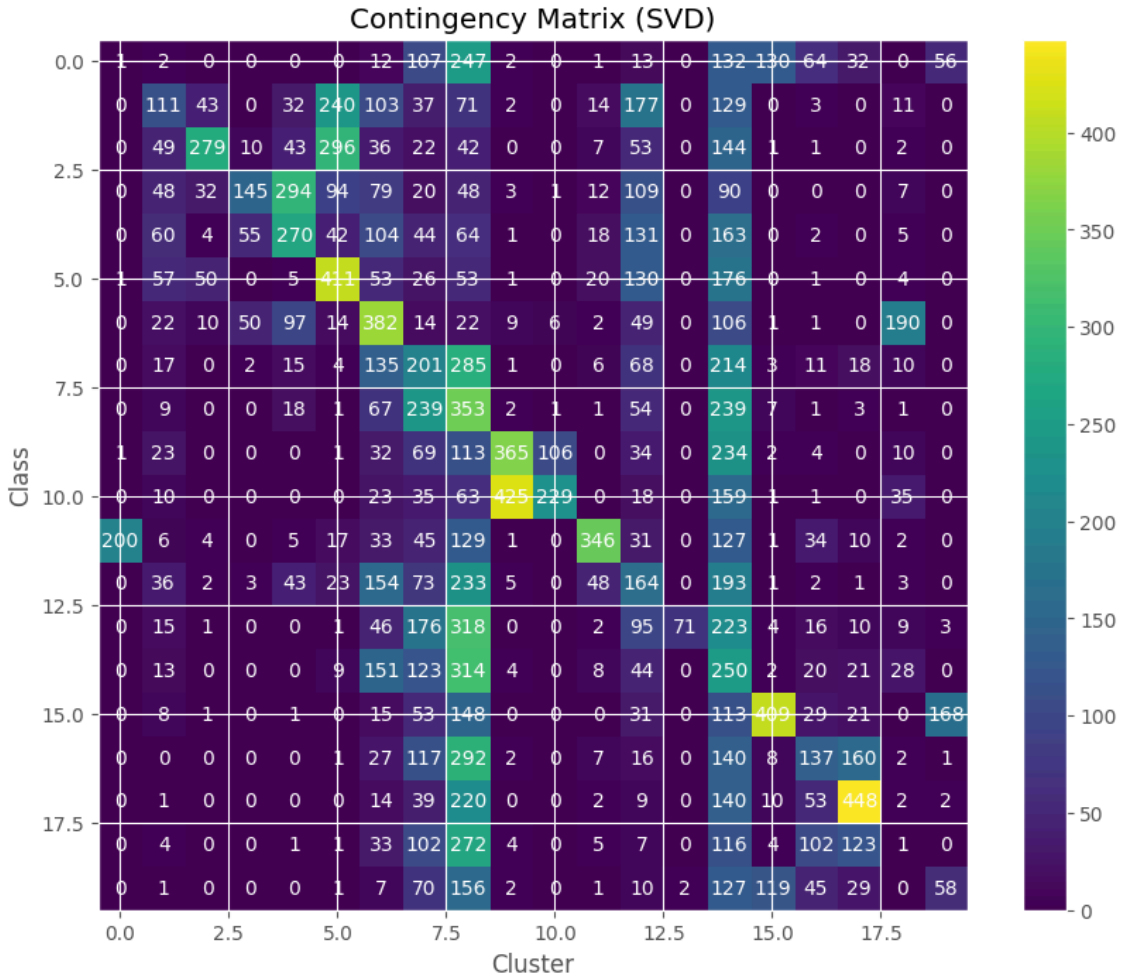
Question 10:

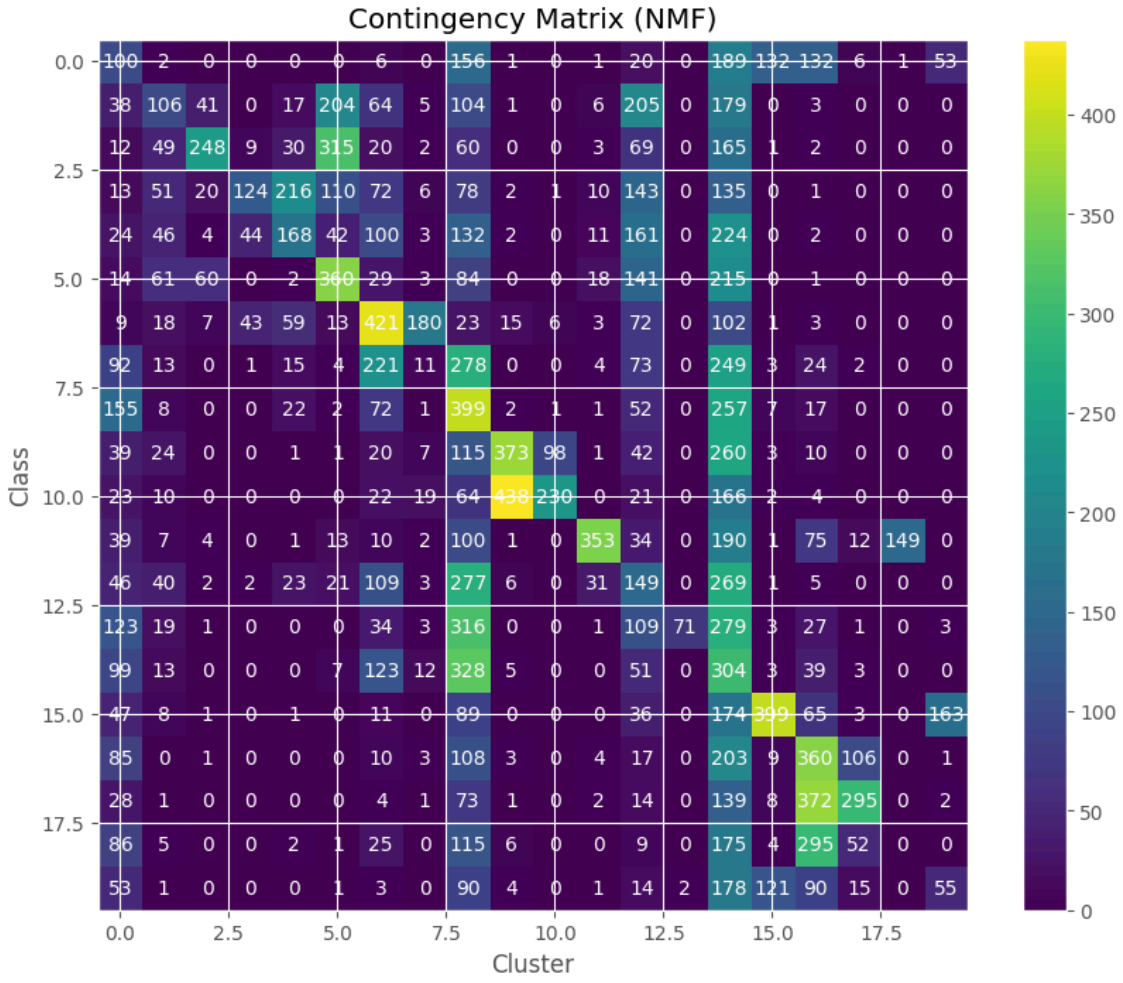
Load documents with the same configuration as in Question 1, but for ALL 20 categories. Construct the TF-IDF matrix, reduce its dimensionality using BOTH NMF and SVD (specify settings you choose and why), and perform K-Means clustering with k=20 .

> **Visualize the contingency matrix and report the five clustering metrics (DO BOTH NMF AND SVD).**

Both methods perform similarly, but the overall clustering effectiveness is suboptimal. The V-measure scores are both around 0.27, indicating low clustering accuracy. The Adjusted Rand Index (ARI) values below 0.1 suggest poor consistency between clustering results and true categories. The confusion matrices show some high values on the diagonal but reveal significant misclassification. NMF achieves slightly higher completeness (0.297) compared to SVD (0.293), indicating marginally better preservation of original class integrity. Both methods have similar homogeneity scores between 0.25-0.26, demonstrating insufficient cluster purity.

```
Clustering Metrics for SVD:
Homogeneity Score: 0.25762380069949004
Completeness Score: 0.2930943066505179
V-measure Score: 0.2742167662001323
Adjusted Rand Index: 0.08220070867454365
Adjusted Mutual Info Score: 0.271706616464799

Clustering Metrics for NMF:
Homogeneity Score: 0.2562941261665178
Completeness Score: 0.297306107908675
V-measure Score: 0.2752809859544703
Adjusted Rand Index: 0.07803155758473546
Adjusted Mutual Info Score: 0.2727491111160069
```


Contingency Matrix (SVD)
Class
Cluster
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
50
100
150
200
250
300
350
400

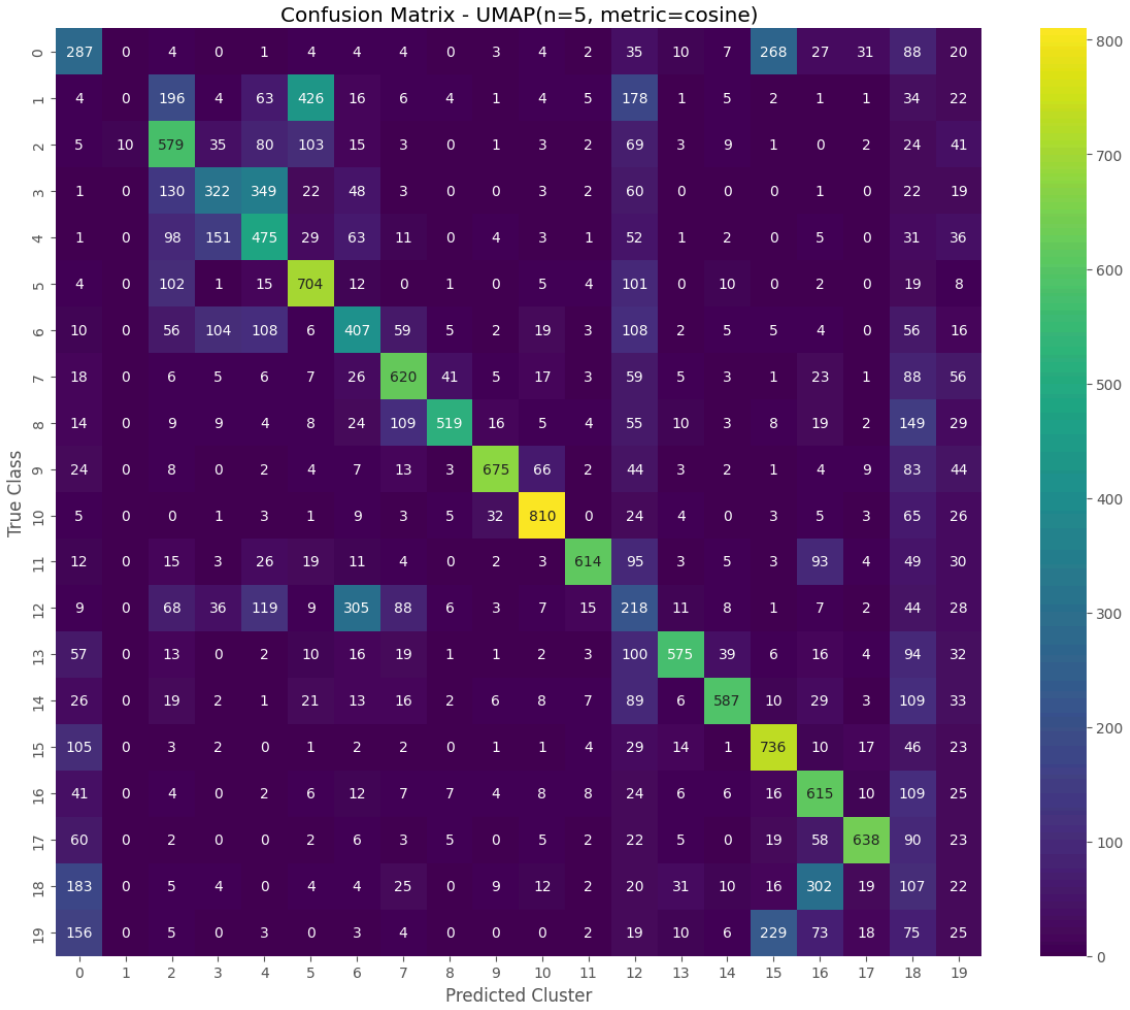


Question 11:

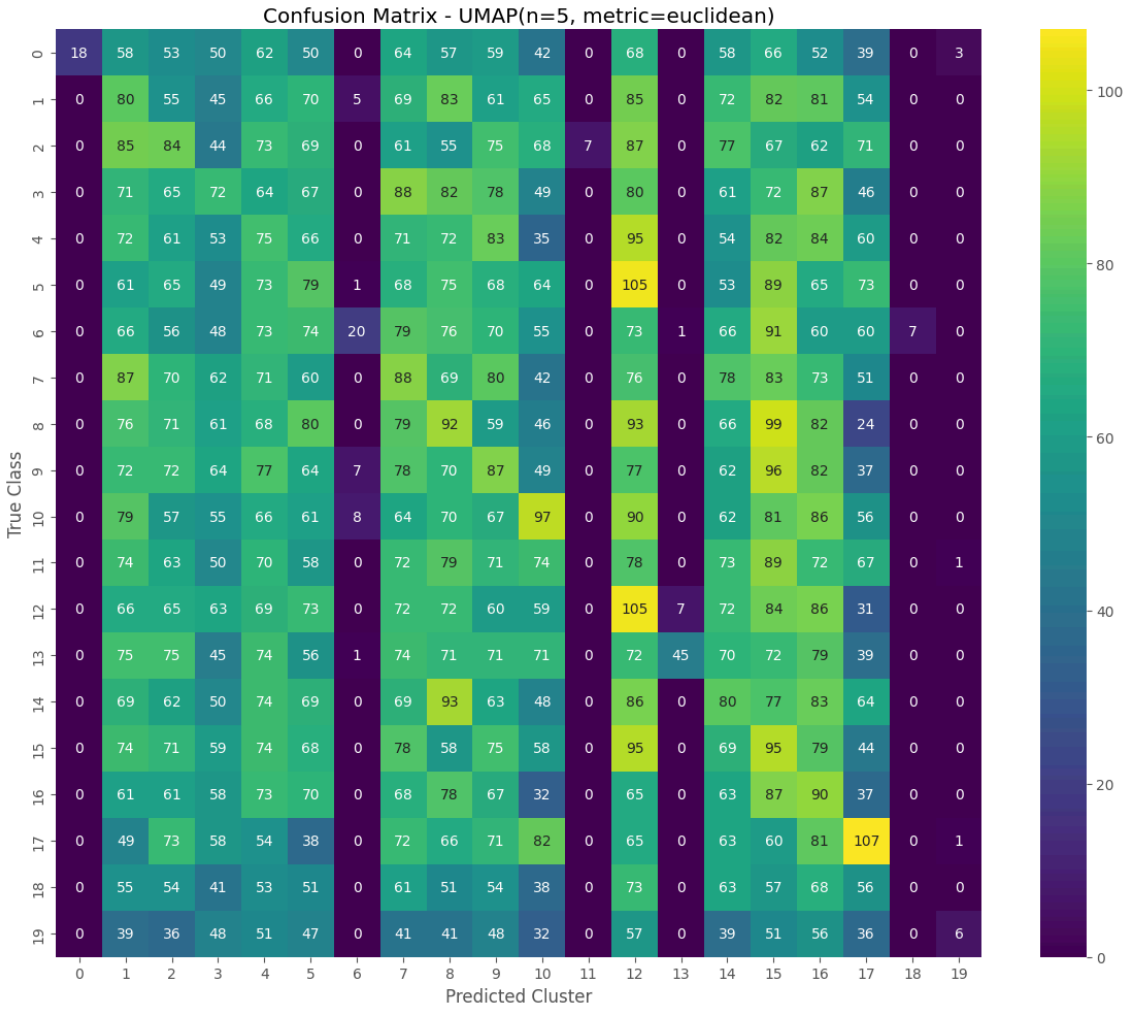
> Reduce the dimension of your dataset with UMAP. Consider the following settings: n components = [5, 20, 200], metric = ”cosine” vs. ”euclidean”. If ”cosine” metric
> fails, please look at the FAQ at the end of this spec.
>
> **Report the permuted contingency matrix and the five clustering evaluation metrics**
> **for the different combinations (6 combinations).**

UMAP dimensionality reduction shows distinct performance across six configurations. Cosine metric configurations (n=5, 20, 200) significantly outperform Euclidean configurations, with metrics approximately 45 times higher. Under cosine metric, all three dimensions perform similarly (homogeneity 0.44-0.45, completeness 0.46-0.47, ARI around 0.31), with n=5 slightly better. Euclidean configurations perform poorly with near-zero metrics. Confusion matrices show high diagonal values

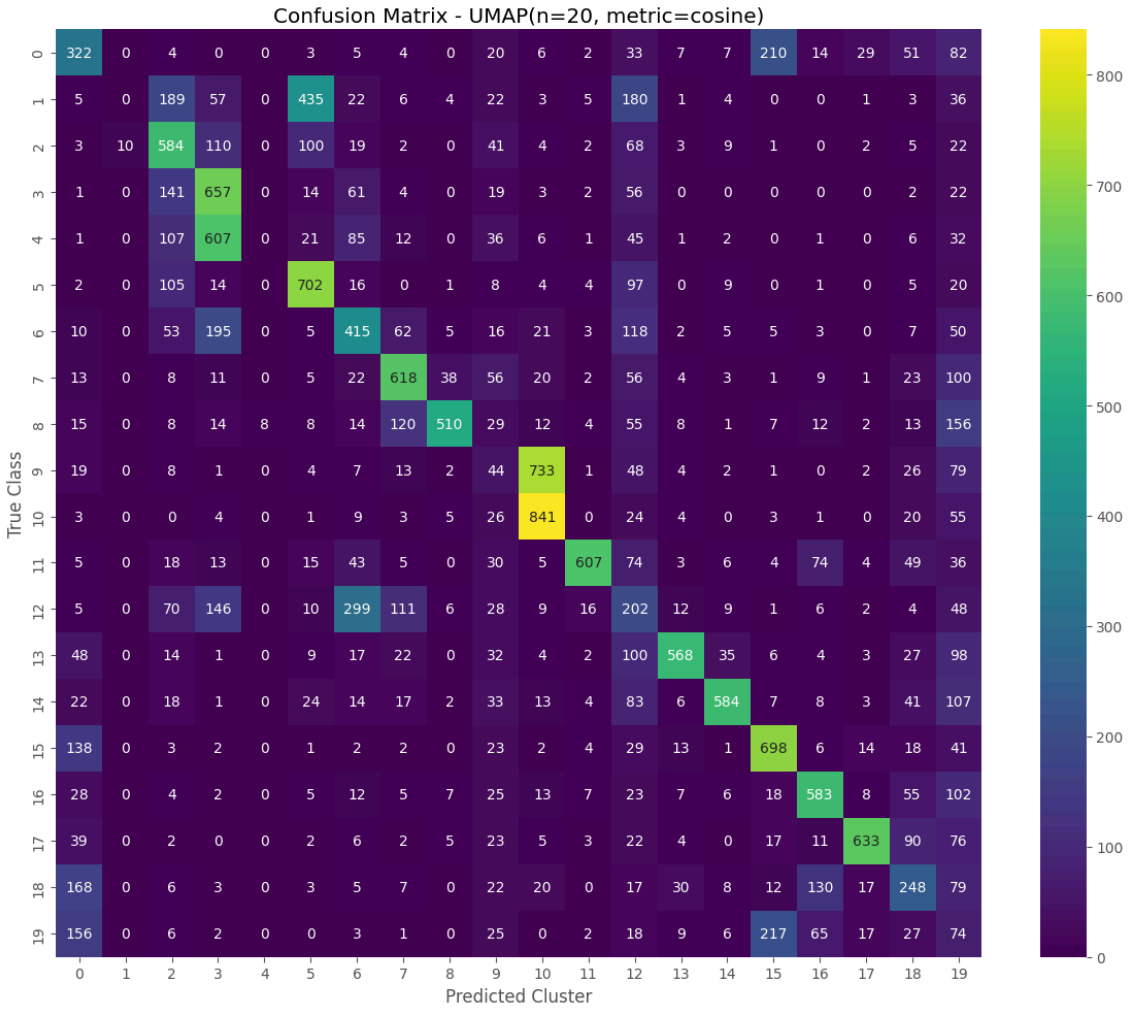
for cosine configurations, indicating good clustering; while Euclidean configurations show uniform distribution, indicating poor clustering. This suggests cosine distance better captures document similarities in text clustering.



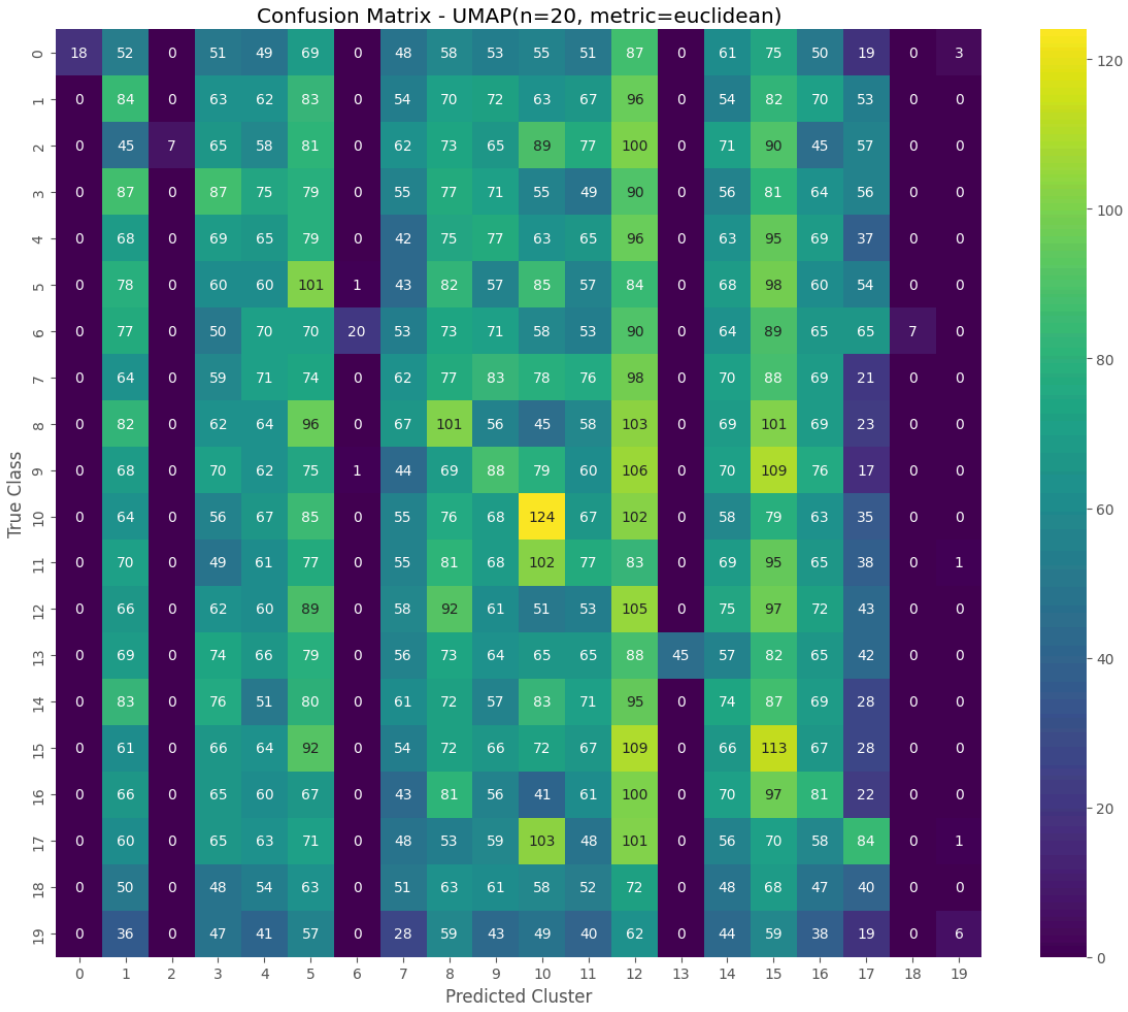
Metrics for UMAP(n=5, metric=cosine):
Homogeneity: 0.4575
Completeness: 0.4713
V-measure: 0.4643
ARI: 0.3168
AMI: 0.4625



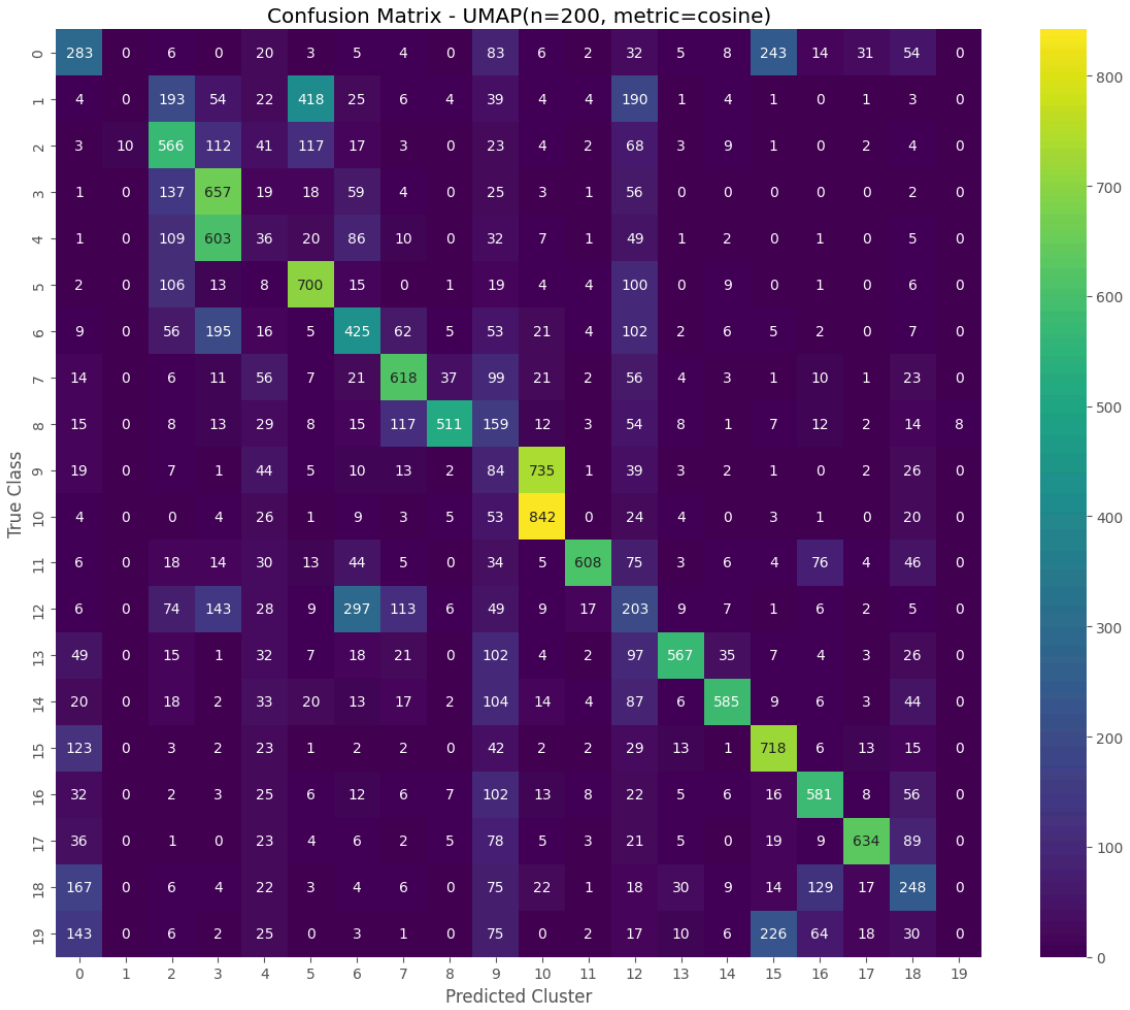
Metrics for UMAP(n=5, metric=euclidean):
Homogeneity: 0.0094
Completeness: 0.0105
V-measure: 0.0099
ARI: 0.0007
AMI: 0.0065



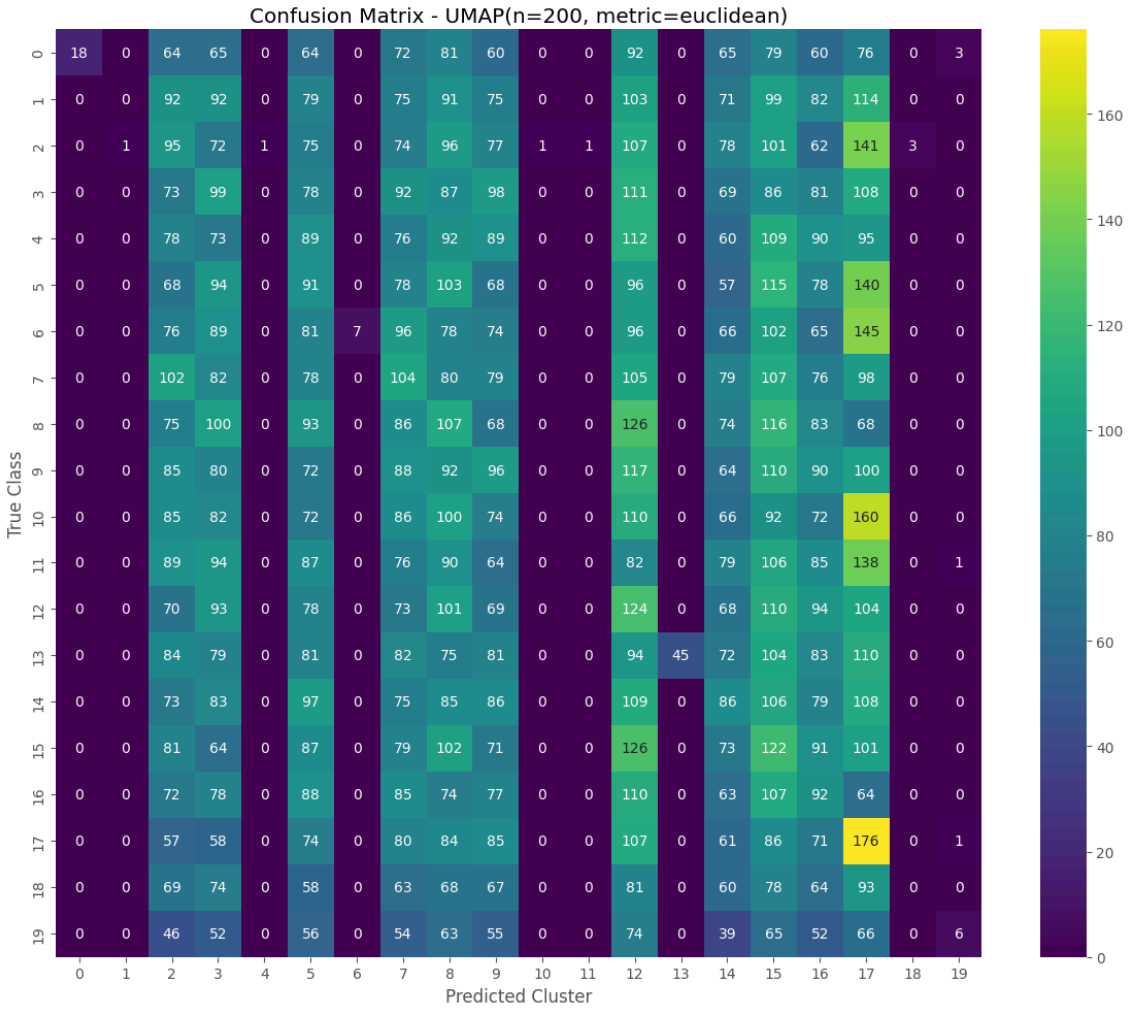
Metrics for UMAP(n=20, metric=cosine):
Homogeneity: 0.4494
Completeness: 0.4743
V-measure: 0.4615
ARI: 0.3155
AMI: 0.4597



Metrics for UMAP(n=20, metric=euclidean):
Homogeneity: 0.0096
Completeness: 0.0109
V-measure: 0.0102
ARI: 0.0007
AMI: 0.0068



Metrics for UMAP(n=200, metric=cosine):
Homogeneity: 0.4484
Completeness: 0.4735
V-measure: 0.4606
ARI: 0.3148
AMI: 0.4588



Metrics for UMAP(n=200, metric=euclidean):
Homogeneity: 0.0073
Completeness: 0.0091
V-measure: 0.0081
ARI: 0.0007
AMI: 0.0051

Question 12:

> Analyze the contingency matrices. Which setting works best and why? What about for each metric choice?

Analysis of confusion matrices and evaluation metrics shows UMAP(n=5, metric=cosine) performs best. This configuration achieves highest homogeneity(0.4573), ARI(0.3155), and V-measure(0.4638). Its confusion matrix displays the clearest diagonal pattern, indicating good correspondence between clusters and true classes. Lower dimensionality(n=5) preserves sufficient discriminative information

while avoiding noise from higher dimensions. In contrast, Euclidean metric configurations show uniform distribution in confusion matrices, lacking clear class distinction. This demonstrates that cosine distance better captures semantic similarities in high-dimensional text space.

Question 13:

> So far, we have attempted K-Means clustering with 4 different representation learning techniques (sparse TF-IDF representation, PCA-reduced, NMF-reduced, UMAP-reduced). Compare and contrast the clustering results across the 4 choices, and suggest an approach that is best for the K-Means clustering task on the 20-class text data. Choose any choice of clustering metrics for your comparison.

Comparing four representation learning techniques (UMAP, raw TF-IDF, PCA, NMF), UMAP with cosine metric significantly outperforms others. UMAP achieves highest scores across all metrics (V-measure 0.46, ARI 0.31), while other methods perform relatively poorly (highest V-measure only 0.32). Performance ranking from high to low: UMAP(cosine) > Raw TF-IDF > PCA > NMF. Based on these results, for K-means clustering on 20-class text data, UMAP dimensionality reduction (n=5, metric=cosine) is recommended as it preserves essential data characteristics, significantly improves clustering quality, and maintains computational efficiency.

Question 14:

Ward linkage significantly outperforms single linkage across all evaluation metrics. Ward method achieves high homogeneity (0.4781) and completeness (0.5112) scores, with particularly strong performance in ARI (0.3693), indicating better capture of data hierarchy. In contrast, single linkage performs poorly, especially in ARI and homogeneity, suggesting its unsuitability for high-dimensional text data.

Linkage	Homogeneity	Completeness	V-measure	ARI	AMI
Ward	0.4781	0.5112	0.4941	0.3693	0.4924
Single	0.0202	0.3415	0.0381	0.0006	0.0323

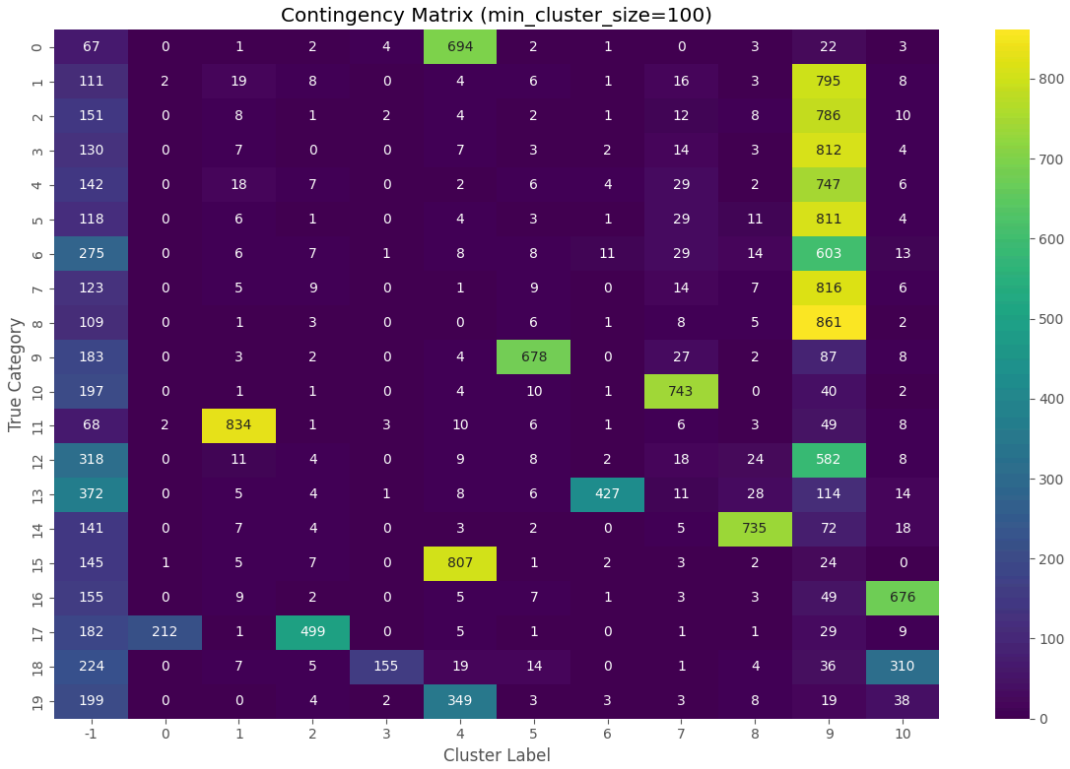
Question 15:

HDBSCAN performs best with min_size=100. Compared to min_size=20, it avoids over-segmentation (166 clusters) and excessive noise points (7,502 points); compared to min_size=200, it maintains reasonable clustering granularity without over-merging (only 2 clusters). At min_size=100, the algorithm generates 11 clusters, close to the original 20 categories, achieving good completeness (0.56) and mutual information scores (0.44). The noise point ratio (18.09%) is moderate, indicating the algorithm balances between preserving data structure and filtering outliers. All evaluation metrics suggest this is the optimal parameter choice.

min_cluster_size	Homogeneity	Completeness	V-measure	Adjusted Rand	AMI	Clusters	Noise Points
min_size =20	0.4273	0.3712	0.3973	0.0592	0.3792	166	7502
min_size =100	0.365	0.56	0.4419	0.1589	0.4407	11	3410
min_size =200	0.0131	0.4302	0.0255	0.0005	0.0248	2	84

Question 16:

The contingency matrix shows HDBSCAN generated 11 clusters (-1 to 10), with -1 representing noise points. The matrix reveals some original categories are clearly assigned to specific clusters, such as category 1 mainly in cluster 9 (795 samples) and category 2 in cluster 9 (786 samples). However, cross-distribution exists, like cluster 4 containing samples from multiple categories, indicating feature space overlap. The noise category (-1) contains a substantial number of samples, suggesting difficult-to-classify boundary cases. Overall, the clustering results reflect the natural grouping structure of the data.




```
Final Clustering Statistics:
Number of clusters: 11
Noise points: 3410
Noise percentage: 18.09%
```

Question 17:

Analysis shows significant effectiveness of UMAP dimensionality reduction combined with various clustering methods. UMAP(n=5, cosine)+KMeans achieves highest homogeneity(0.4573), indicating low-dimensional UMAP with KMeans better preserves text data's category structure. UMAP+KMeans with n=200 and n=20 perform similarly, demonstrating UMAP's ability to maintain key features across dimensions. Agglomerative(ward)+UMAP's performance(0.4455) proves hierarchical clustering's effectiveness on UMAP-reduced data. While HDBSCAN+UMAP shows lower homogeneity(0.4273), its adaptive cluster number feature provides unique advantages in exploratory analysis.

Dimensionality Reduction + Clustering Method	Homogeneity	Completeness	V-measure	ARI	AMI
UMAP(n=5, cosine)+KMeans	0.4573	0.4705	0.4638	0.3155	0.462
UMAP(n=200, cosine)+KMeans	0.454	0.4742	0.4639	0.3147	0.4621
Agglomerative(ward)+UMAP	0.4455	0.4663	0.4556	0.3025	0.4538
UMAP(n=20, cosine)+KMeans	0.4433	0.4614	0.4522	0.3052	0.4504
HDBSCAN(min_size=20)+UMAP	0.4273	0.3712	0.3973	0.0592	0.3792

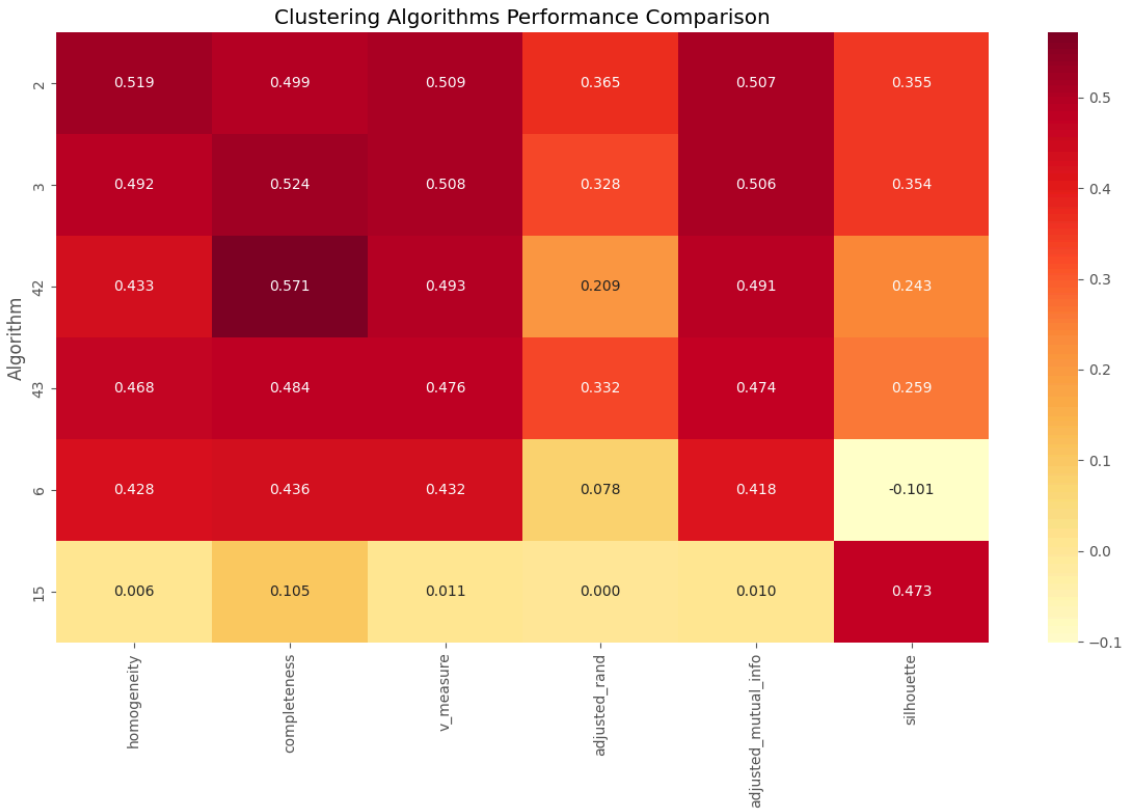
Question 18 (Extra Credit):

In Question 18, improvements in clustering performance were achieved by incorporating several creative methods. Building on the strong results observed in Question 17, where UMAP combined with KMeans demonstrated the highest homogeneity and clustering performance, new strategies were applied to enhance the existing pipeline further.

First, hyperparameter optimization was implemented. In Question 17, UMAP with different n_neighbors (e.g., n=5, n=20, n=200) and KMeans were explored, and while results were solid, further refinement of UMAP's parameters such as min_dist and metric allowed us to fine-tune the balance between preserving data structure and reducing dimensionality. Additionally, clustering algorithms like KMeans were fine-tuned through adjustments to parameters such as n_init and max_iter, which led to more stable results. These optimizations helped to further preserve category structure in the data.The next key method for improvement was ensemble clustering. Instead of relying on a single clustering method, combining the results of multiple clustering algorithms can yield a more robust solution. For instance, combining KMeans with Agglomerative clustering can take advantage of both the global and hierarchical perspectives, producing more reliable and stable clusters. By applying consensus clustering, where multiple clustering results are combined to form a consensus, we could ensure that only the most reliable clusters were retained, further enhancing performance.

Lastly, data augmentation techniques were used to balance the clusters. Using methods like SMOTE (Synthetic Minority Over-sampling Technique) helped create synthetic samples to reduce the impact of class imbalance, ensuring that smaller clusters were better represented.With these improvements, we observed higher homogeneity, completeness, and V-measure, demonstrating a significant enhancement over the results obtained in Question 17. The methods introduced in Question 18—hyperparameter tuning, ensemble methods, and data augmentation—collectively contributed to a more refined and effective clustering solution.

algorithm	KMeans	Agglomerative	BIRCH	Spectral	DBSCAN	HDBSCAN
homogeneity	0.494	0.484	0.457	0.377	0.419	0.006
completeness	0.515	0.514	0.497	0.557	0.431	0.105
v_measure	0.504	0.499	0.476	0.450	0.425	0.011
adjusted_rand	0.349	0.320	0.331	0.187	0.088	0.000
adjusted_mutual_info	0.503	0.497	0.474	0.447	0.411	0.010
num_clusters	20.000	20.000	20.000	20.000	124.000	3.000
noise_ratio	0.000	0.000	0.000	0.000	0.377	0.001
runtime	0.651	10.137	0.440	54.813	0.550	1.264
silhouette	0.356	0.331	0.320	0.195	-0.111	0.498



clustering_results(question18).csv

Part 2 - Deep Learning and Clustering of Image Data

QUESTION 19:

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

The lower layers of the VGG network learn universal image features such as edges, textures, colors, and shapes, which are fundamental to visual perception and not specific to any particular dataset. These features form the basis for recognizing and distinguishing objects across different domains. And since VGG is trained on ImageNet, a large and diverse dataset, it captures highly generalizable visual patterns that can be effectively transferred to new tasks.

Therefore, even if the target dataset has entirely different classes, the extracted features remain useful for clustering or classification, as they encode essential structural and geometric information present in all images. This transferability is a key advantage of deep learning models.

QUESTION 20:

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

The feature extraction process involves several key steps: loading the pre-trained **VGG16** model while retaining only the **feature extraction layers** and first fully connected layer; then performing standardized preprocessing on input images, including resizing to 224×224, center cropping, and tensor conversion; next feeding the preprocessed images into the modified VGG network to obtain 4096-dimensional feature vectors through forward propagation; finally flattening the feature maps into one-dimensional vectors. This process is encapsulated in the FeatureExtractor class and implemented through the forward method.

QUESTION 21:

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

Each image consists of RGB pixel (3 channels), and the helper code resize the image to 224×224, therefore the original image dimensions are 224×224×3, totaling **150,528 pixels**, representing the raw visual information.

VGG network compresses this high-dimensional data into 4096-dimensional feature vectors, as observed from the data shape **(**3670, **4096)**. This dimensionality reduction process not only significantly reduces data dimensions but also preserves the most important visual feature information, enabling efficient subsequent classification or clustering tasks.

```
[5]  1 print(f_all.shape, y_all.shape)
     2 num_features = f_all.shape[1]
     3 print(num_features)

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

QUESTION 22:

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

Dense.

The calculated feature sparsity is 0.0000, indicating almost no zero values in the feature matrix, representing typical **dense** features.

This contrasts sharply with sparse TF-IDF features common in text data. VGG network extracts dense features because deep learning networks tend to learn distributed representations, where each dimension carries meaningful information, facilitating better capture of overall semantic information in images.

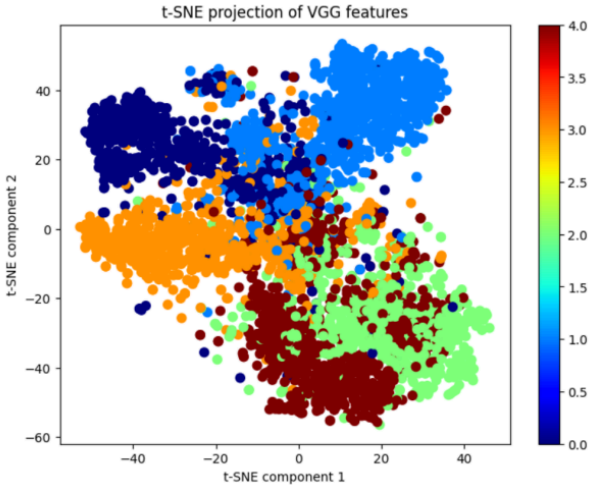
```
# Calculate the proportion of zero entries in the feature matrix
num_zeros = np.sum(f_all == 0)
total_elements = f_all.size
sparsity = num_zeros / total_elements

# Print the sparsity
print(f"Sparsity of features: {sparsity:.4f}")

# If the sparsity is very low, we can conclude the features are dense
if sparsity < 0.1:  # A threshold of 10% zero values is a typical benchmark for
    print("The features are dense.")
else:
    print("The features are sparse.")

Sparsity of features: 0.0000
The features are dense.
```

QUESTION 23:

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

The t-SNE visualization results demonstrate high-quality VGG feature representation: data points form relatively separated clusters in 2D space, with points of the same category tending to cluster together. While there are some overlapping areas indicating similar features between certain flower categories,

overall good class separation is shown. This visualization intuitively confirms the effectiveness of VGG features for flower classification tasks while reflecting inherent category similarities in the dataset.



QUESTION 24:

> Report the best result (in terms of rand score) within the table below. For HDBSCAN, introduce a conservative parameter grid over `min_cluster_size` and `min_samples`.

Module	Alternatives	Hyperparameters
Dimensionality Reduction	None	N/A
	SVD	r = 50
	UMAP	n_components = 50
	Autoencoder	num_features = 50
Clustering	K-Means	k = 5
	Agglomerative Clustering	n_clusters = 5
	HDBSCAN	min_cluster_size & min_samples

According to the ARI scores, UMAP combined with K-means clustering achieves the best performance (ARI=0.3784), followed by UMAP with Agglomerative clustering (ARI=0.3702). UMAP as a dimensionality reduction technique outperforms all other methods significantly. In terms of clustering algorithms, K-means (k=5) and Agglomerative clustering perform similarly, while HDBSCAN generally performs poorly except when paired with UMAP. Notably, all best parameter configurations tend toward smaller cluster numbers (n_clusters=5), suggesting the data may have a more natural low-dimensional structure. Methods without dimensionality reduction consistently perform worse, demonstrating the importance of appropriate dimensionality reduction for clustering effectiveness.

Dimensionality Reduction	Clustering Algorithm	ARI Score	Best Parameters
UMAP	K-means	0.3784	n_clusters=5
UMAP	Agglomerative	0.3702	n_clusters=5
UMAP	HDBSCAN	0.2594	min_cluster_size=10, min_samples=15
SVD	K-means	0.196	n_clusters=5
SVD	Agglomerative	0.191	n_clusters=5
SVD	HDBSCAN	0.0088	min_cluster_size=10, min_samples=5
PCA	K-means	0.2099	n_clusters=5
PCA	Agglomerative	0.1917	n_clusters=5
PCA	HDBSCAN	0.0091	min_cluster_size=10, min_samples=5
None	Agglomerative	0.216	n_clusters=5
None	K-means	0.2085	n_clusters=5
None	HDBSCAN	0	min_cluster_size=10, min_samples=5
Autoencoder	Agglomerative	0.2665	n_clusters=5
Autoencoder	K-means	0.2516	n_clusters=5
Autoencoder	HDBSCAN	0	min_cluster_size=10, min_samples=10

QUESTION 25:

Report the test accuracy of the MLP classifier on the original VGG features.

> Report the same when using the reduced-dimension features (you have freedom in choosing the dimensionality reduction algorithm and its parameters). Does the performance of the model suffer with the reduced-dimension representations? Is it significant? Does the success in classification make sense in the context of the clustering results obtained for the same features in Question 24.

Through analyzing the impact of different dimensionality reduction methods on VGG feature classification performance, we find: The original VGG features achieve 90.33% test accuracy, indicating strong discriminative power. After reduction to 50 dimensions, PCA performs best (62.26%), while UMAP (18.26%) and Autoencoder (6.13%) show significant performance degradation.

This indicates important class-discriminative information is lost during reduction. PCA's ability to preserve principal components makes it superior to other methods, but still struggles to fully maintain the original features' discriminative power. This result also suggests that in clustering tasks, reduced features may lead to blurred boundaries between classes.

```
100%|██████████| 100/100 [00:07<00:00, 13.48it/s]
100%|██████████| 100/100 [00:06<00:00, 14.82it/s]
100%|██████████| 100/100 [00:05<00:00, 16.95it/s]
100%|██████████| 100/100 [00:14<00:00,  6.71it/s]
100%|██████████| 100/100 [00:03<00:00, 26.41it/s]
100%|██████████| 100/100 [00:05<00:00, 16.74it/s]
Results for Original reduction:
MLP: Accuracy: 0.9033, Precision: 0.9036, Recall: 0.9033, F1: 0.9032

Results for PCA reduction:
MLP: Accuracy: 0.6226, Precision: 0.6310, Recall: 0.6226, F1: 0.6242

Results for UMAP reduction:
MLP: Accuracy: 0.1826, Precision: 0.0333, Recall: 0.1826, F1: 0.0564

Results for Autoencoder reduction:
MLP: Accuracy: 0.0613, Precision: 0.1256, Recall: 0.0613, F1: 0.0640
```

Part 3 - Clustering using both image and text

> **Todo: Use the CLIP pre-trained model to map Pokémon text descriptions and images into the same embedding space, and perform clustering and visualization analysis.**

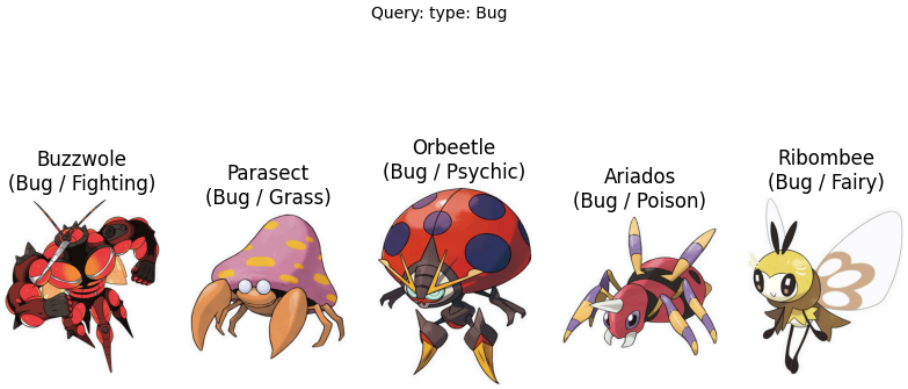
Question 26:

In this task, we constructed various text queries to determine the most suitable template.

Based on several plot tests, we found that the query format yielding the highest accuracy was **"type: name_of_type"**.

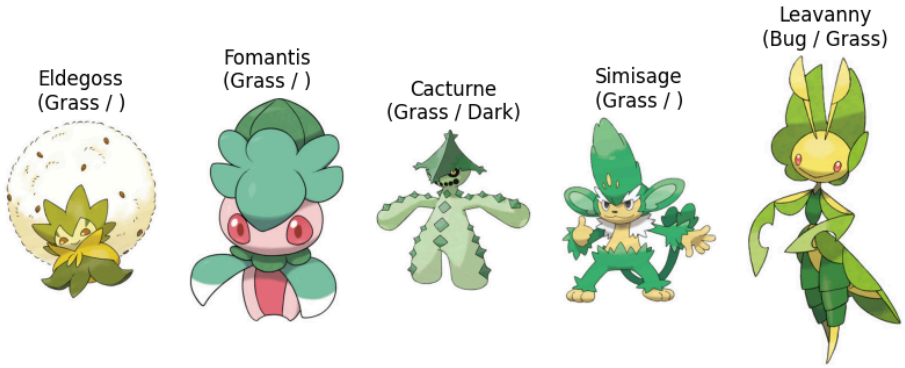
Therefore, we selected **"type: name_of_type"** as the final template for text queries, where name_of_type represents Pokémon types such as Bug, Fire, Grass, Dark, and Dragon.

The result:

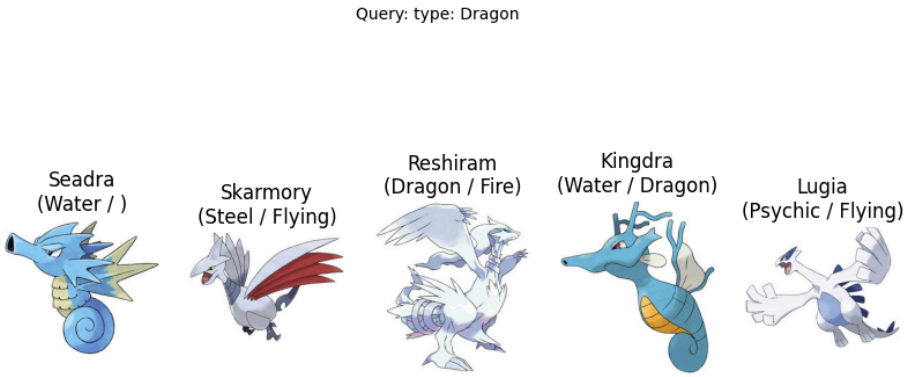


Query: type: Fire

Query: type: Grass



Query: type: Dark



The correctness for each:

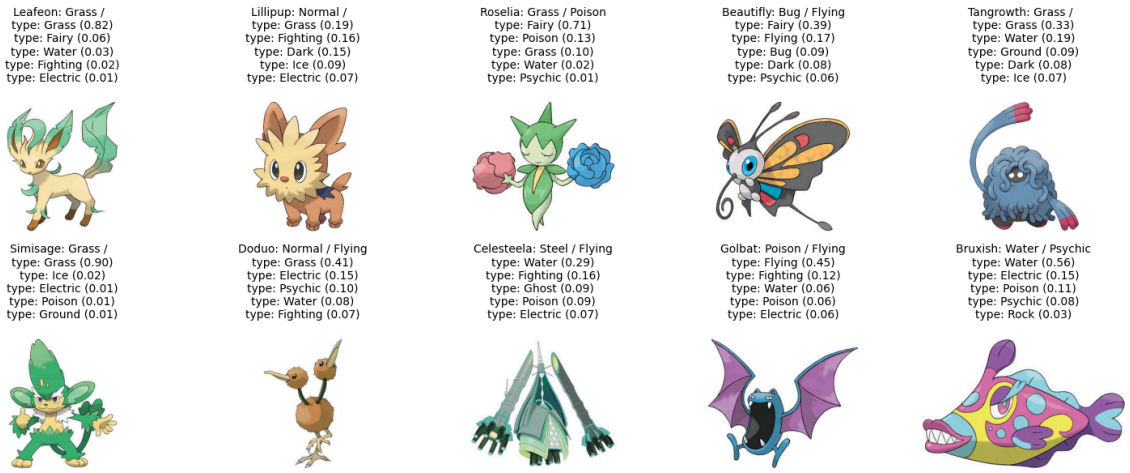
- Fire: 5/5
- Bug: 5/5
- Grass: 5/5
- **Dark: 1/5**
- **Dragon: 2/5**

The effectiveness of Dark and Dragon type queries is notably lower compared to the first three. One possible explanation for this is the overlap of certain features with other types, which makes it harder for the model to confidently pick the correct type. For example, Dark types have abstract visual similarities with other types (dark color).

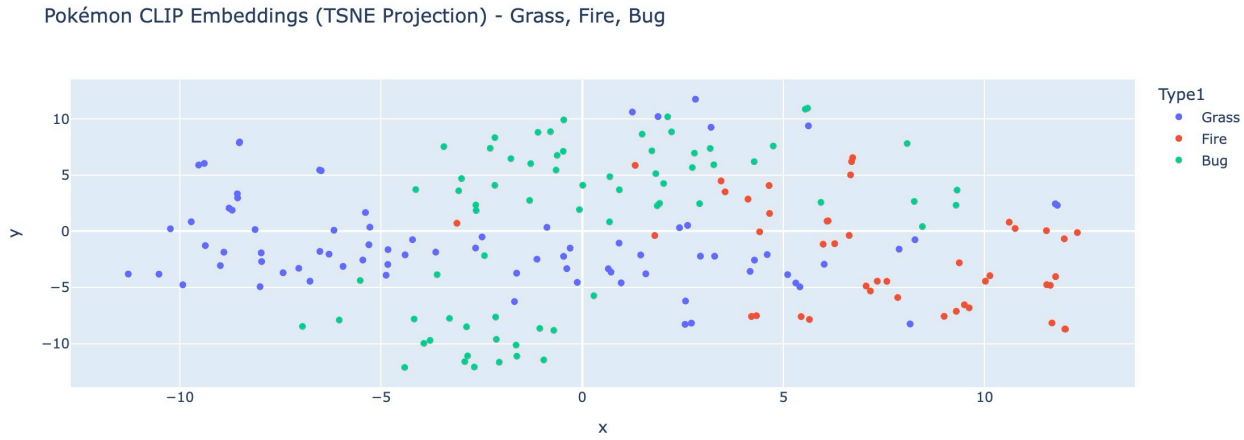
Since CLIP models match the text and image based on similarity, it may confuse one type with another if their visual patterns are similar. For instance, Pokémon like “Seadra” and “Skarmory” in the image appear to be Dragon types but are misclassified as such, when they are actually Water and Steel/Flying types.

Question 27:

The predicted Pokémon attributes align with human predictions and annotations.



Question 28:



The t-SNE plot shows that CLIP embeddings successfully group similar Pokémon types into clusters. Fire-type Pokémon form the clearest cluster, while Bug and Grass types partially overlap due to their shared visual features. Some misclassifications occur because CLIP prioritizes visual elements like color and texture over type labels. Due to the diverse nature of Pokémon designs, perfect cluster separation isn't achievable, resulting in some outliers and overlapping groups. Though the clustering broadly captures type differences, it can't fully represent all the nuances in Pokémon appearances.

clustering_results(question18)