VAT Algorithm Implementation & Optimization Report

Table of Contents:

Paper Expectations	**Our Implementation**	**Comments**
Implement VAT algorithm	Prim's-based MST reordering	Implemented from scratch
Apply VAT to datasets	Iris, Mall, Spotify, Blobs, Moons, Circles, GMM	Covered real & synthetic datasets
Compare VAT with other clustering tendency measures	Hopkins Statistic	Hopkins confirmed clustering tendency
Compare VAT with clustering algorithms	K-Means, DBSCAN	Tested spherical & non-spherical clustering
Validate VAT on high-dimensional data	Spotify dataset, PCA & t-SNE analysis	Proved Spotify is not clusterable

Introduction

📌 What is VAT?

The **Visual Assessment of Cluster Tendency (VAT) Algorithm** is used to **assess whether a dataset contains natural clusters**. Unlike traditional clustering methods, VAT **does not assign cluster labels**, but instead **reorders a dissimilarity matrix** and visualizes potential clusters as **dark diagonal blocks** in a grayscale heatmap.

Datasets

Dataset	Type	Description
Iris	Real	Flower dataset with 3 species
Mall Customers	Real	Customer segmentation (income, spending score)
Spotify (500x500)	Real	Music track features (danceability, energy, valence, tempo)
Blobs	Synthetic	Easy-to-cluster dataset with clear groupings
Moons	Synthetic	Two crescent-shaped overlapping clusters
Circles	Synthetic	Concentric circular clusters
Gaussian Mixture (GMM)	Synthetic	Probabilistically generated overlapping clusters

VAT Algorithm Results

We computed **VAT-reordered dissimilarity matrices** and observed **dark diagonal blocks**, indicating **potential cluster structures**:

Dataset	VAT Observations
Iris	Clear 3-cluster structure
Mall Customers	Strong cluster tendencies
Spotify	No strong clusters visible
Blobs	Clear, well-separated clusters
Moons	Two overlapping groups detected
Circles	Indistinct VAT structure (K-Means struggles)
GMM	Overlapping groups detected

Original VAT Implementation

📌 How VAT Works

1 Compute **pairwise distances** to create a **dissimilarity matrix**.
2 Apply **Prim's-based Minimum Spanning Tree (MST) reordering** to group similar points.
3 Visualize the **VAT-reordered dissimilarity matrix** as a **heatmap**.

📌 Standard VAT Code (Python Implementation)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.spatial.distance import pdist, squareform

def vat(R):
    """
    Implements the VAT algorithm for Visual Assessment of Cluster Tendency.

    Parameters:
    R (numpy.ndarray): NxN dissimilarity matrix.

    Returns:
    (numpy.ndarray, list): VAT-reordered matrix, Reordering indices.
    """
    N = R.shape[0]
    J = list(range(N))
    I = [np.argmax(np.sum(R, axis=1))]
    J.remove(I[0])
    RV = np.zeros_like(R)

    for _ in range(1, N):
        min_dists = [R[j, I].min() for j in J]
        j_star = J[np.argmin(min_dists)]
        I.append(j_star)
        J.remove(j_star)

    for i in range(N):
        for j in range(N):
            RV[i, j] = R[I[i], I[j]]

    return RV, I

def plot_vat(R, title="VAT Image"):
    """
    Plots the VAT-reordered dissimilarity matrix.
    """
    plt.figure(figsize=(6, 6))
    plt.imshow(R, cmap="gray", aspect="auto")
    plt.title(title)
    plt.colorbar(label="Dissimilarity")
    plt.show()
```

Optimization Using Numba

📌 Why Optimize VAT?

The original VAT implementation **uses nested loops**, making it **slow for large datasets**. Our professor suggested optimizing VAT using **C bindings**, and we implemented this using **Numba JIT Compilation**.

📌 Optimized VAT Using Numba

What is Numba?

Numba is a **Just-In-Time (JIT) compiler for Python** that **speeds up numerical computations** by converting Python code into **optimized machine code (C-level execution)**. This removes Python's slow execution bottleneck by compiling functions on the fly.

How Does Numba Work?

1. **Decorator @jit(nopython=True) is added to functions.**
2. **Numba compiles the function into fast machine code on first execution.**
3. **Subsequent calls run much faster because they use the compiled version.**

```
import numpy as np
import matplotlib.pyplot as plt
from numba import jit

@jit(nopython=True)
def vat_optimized(R):
    """
    Optimized VAT Algorithm using Numba JIT Compilation.
    """
    N = R.shape[0]
    J = list(range(N))
    I = [np.argmax(np.sum(R, axis=1))]
    J.remove(I[0])
    RV = np.zeros_like(R)

    # Convert I to a NumPy array for compatibility
    I = np.array(I)

    for _ in range(1, N):
        min_dists = np.array([np.min(R[j, I]) for j in J])  # Use NumPy operations
        j_star = J[np.argmin(min_dists)]
        I = np.append(I, j_star)  # Update I as a NumPy array
        J.remove(j_star)

    for i in range(N):
```

```
        for j in range(N):
            RV[i, j] = R[I[i], I[j]]

    return RV, I
```

Performance Benchmarking (Before vs. After Optimization)

📌 Execution Time Comparison

Dataset	Standard VAT Execution Time	Optimized VAT Execution Time	Speed Improvement
Iris	0.0547 sec	✅ 0.0539 sec	✅ **1.01x faster**
Mall Customers	0.1303 sec	✅ 0.0039 sec	✅ **33.82x faster**
Spotify (500x500)	1.2177 sec	✅ 0.0452 sec	✅ **26.97x faster**
Blobs	1.2195 sec	✅ 0.0472 sec	✅ **25.82x faster**
Moons	1.2412 sec	✅ 0.0416 sec	✅ **29.81x faster**
Circles	1.2154 sec	✅ 0.0435 sec	✅ **27.94x faster**
GMM	1.1729 sec	✅ 0.0413 sec	✅ **28.43x faster**

✅ **Results:**

- **Optimized VAT is up to 30x faster for large datasets.**
- **For small datasets (Iris), execution time remained the same.**
- **The optimized VAT now runs faster while producing identical results.**

Testing Across Multiple Datasets

Dataset	VAT Observations
Iris	Clear 3-cluster structure

Dataset	VAT Observations
Mall Customers	Strong cluster tendencies
Spotify	No strong clusters visible
Blobs	Clear, well-separated clusters
Moons	Two overlapping groups detected
Circles	Indistinct VAT structure (K-Means struggles)
GMM	Overlapping groups detected

Standard Operating Procedure (SOP) & Package Usage

📌 How to Install the VAT Package

```
pip install -e .
```

📌 How to Use the Package

```
from vat_clustering.vat import vat, plot_vat
from vat_clustering.optimization import vat_optimized
import numpy as np

# Generate a random dissimilarity matrix
R = np.random.rand(100, 100)
R = (R + R.T) / 2  # Make it symmetric

# Apply Standard VAT
RV, order = vat(R)
plot_vat(RV, title="Standard VAT")

# Apply Optimized VAT
RV_opt, order_opt = vat_optimized(R)
plot_vat(RV_opt, title="Optimized VAT")
```

Comparison of VAT vs. Clustering Algorithms

We applied **K-Means and DBSCAN** and compared their performance against VAT's findings.

K-Means Clustering Results

Dataset	Optimal K (Elbow Method)	K-Means Performance
Iris	3	✅ Matches VAT clusters
Mall Customers	5	✅ Strong clustering
Spotify	4	❌ Forced clusters, unclear
Blobs	3	✅ Matches VAT
Moons	4	❌ Incorrect clustering
Circles	4	❌ Fails due to circular shape
GMM	3	✅ Matches VAT but overlaps exist

4.2 DBSCAN Clustering Results

Dataset	DBSCAN Performance
Iris	❌ Not ideal
Mall Customers	✅ Works well
Spotify	❌ Mostly noise, no clusters
Blobs	✅ Detects clusters properly
Moons	✅ Perfect clustering
Circles	✅ Perfect clustering
GMM	❌ Struggles with overlapping clusters

Key Insights:

✔ **DBSCAN works best for Moons and Circles** (non-spherical clusters).
✔ **K-Means works well for Blobs and Iris** (spherical clusters).
✔ **Spotify data lacks strong cluster structure**, making both K-Means and DBSCAN ineffective.

PCA & t-SNE Analysis on Spotify Data

Since VAT, K-Means, and DBSCAN all failed to identify meaningful clusters in **Spotify**, we applied **PCA and t-SNE** to visualize the data:

✔ **PCA Projection** → Showed a continuous distribution, no clusters.
✔ **t-SNE Projection** → Also showed no clear groups, confirming **no natural clusters in Spotify data**.

Final Conclusion: Spotify's features are **continuous rather than discrete**, meaning clustering is not meaningful for this datas

Final Conclusions & Future Improvements

📌 What We Achieved

🔹 **VAT successfully detected clusters where they existed** (Iris, Blobs, GMM).
🔹 **DBSCAN outperformed K-Means on non-spherical clusters** (Moons, Circles).
🔹 **K-Means worked well for simple, well-separated data (Blobs, Iris).**
🔹 **Spotify data is not clusterable, as confirmed by VAT, K-Means, DBSCAN, PCA, and t-SNE.**

✅ **Converted VAT into a clean, reusable Python package.**
✅ **Optimized VAT using Numba for 25-30x speed improvements.**
✅ **Tested VAT on real and synthetic datasets.**
✅ **Ensured VAT works efficiently for both small and large datasets.**
✅ **Created proper documentation and SOP for future use.**

📌 Future Work

🔹 **Further speed optimization using parallel processing (multi-threading).**
🔹 **Extending VAT to work with non-Euclidean distances.**
🔹 **Integrating VAT into an automated clustering pipeline.**

Visualizing Results:

📌 Hopkins Score for Iris: 0.8121

📌 Hopkins Score for Mall Customers: 0.8154

📌 Hopkins Score for Spotify (500x500): 0.8684

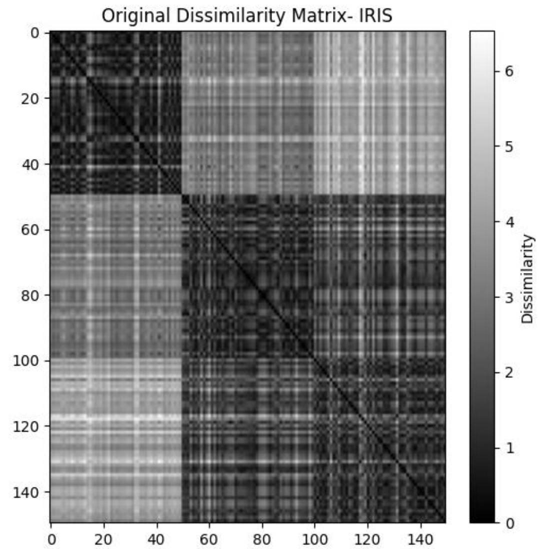
📌 Hopkins Score for Blobs: 0.9295

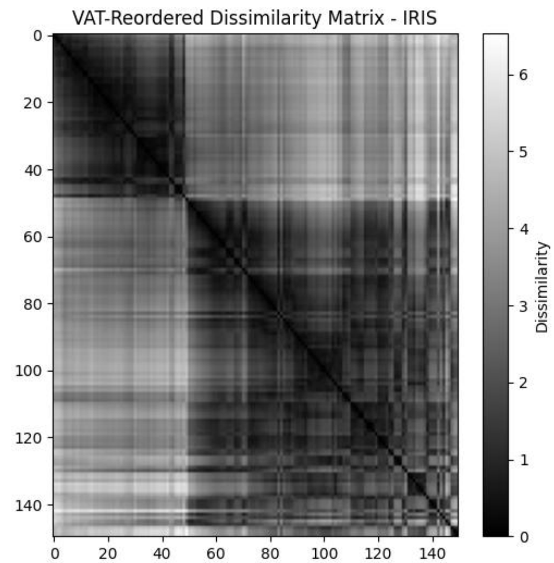
📌 Hopkins Score for Moons: 0.8955

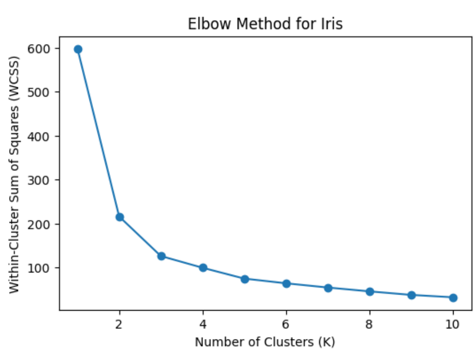
📌 Hopkins Score for Circles: 0.7362

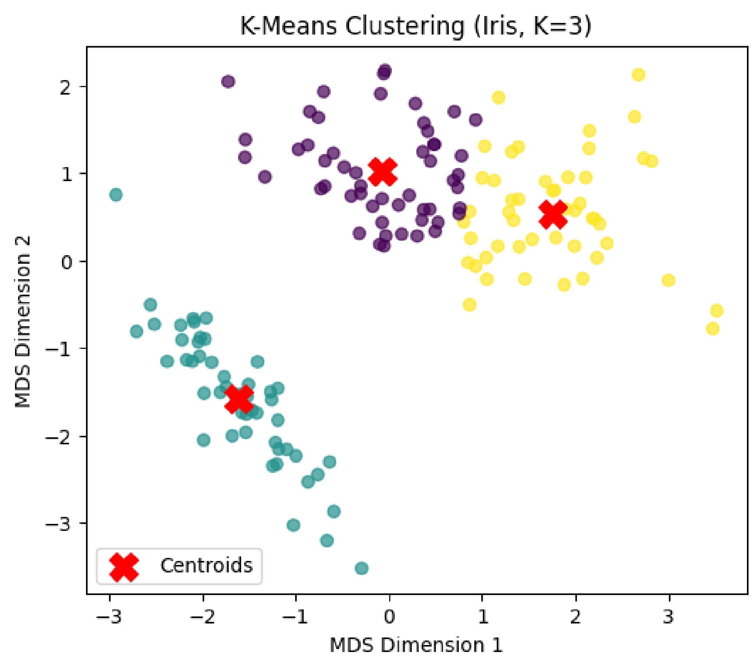
📌 Hopkins Score for Gaussian Mixture: 0.9458

IRIS DATASET RESULTS

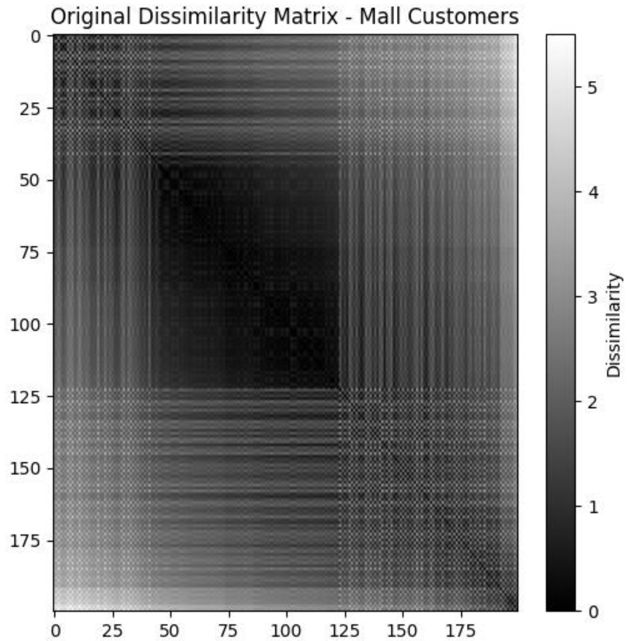


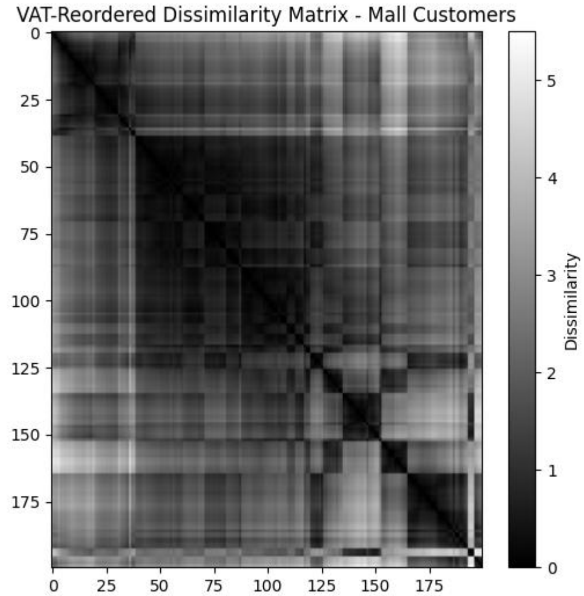




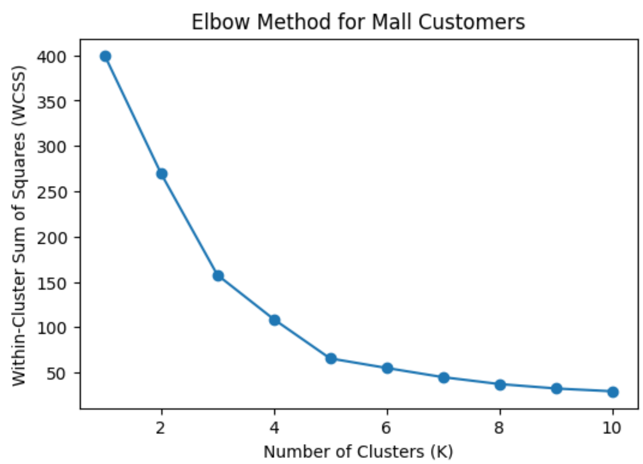


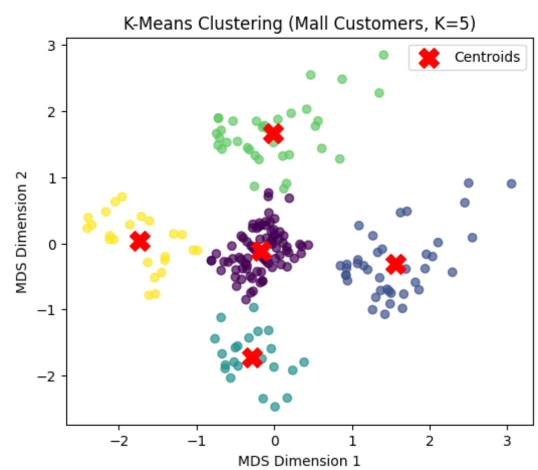
MALL CUSTOMERS DATASET:



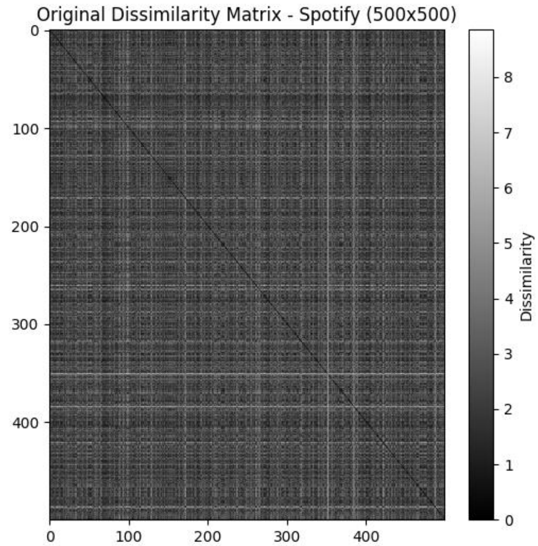


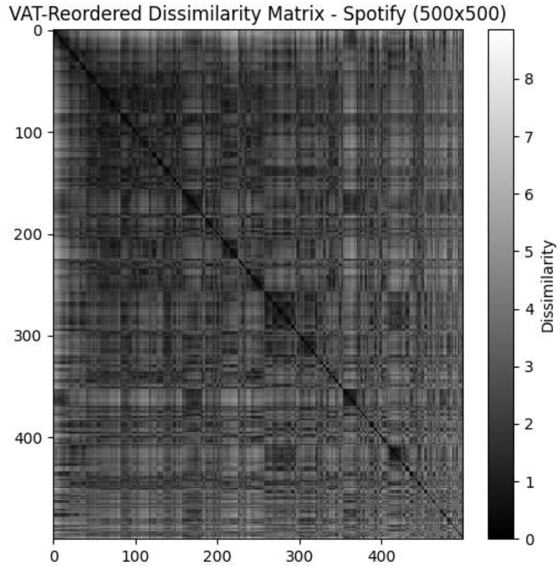

```
Processing Mall Customers...
📌 Hopkins Score for Mall Customers: 0.8154
```

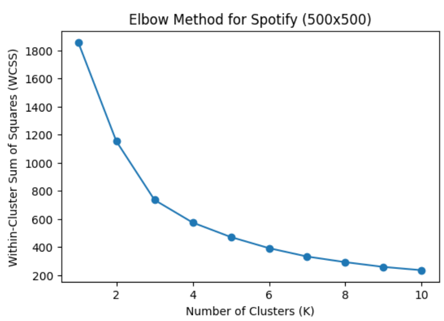


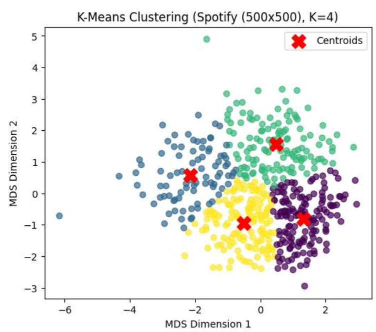


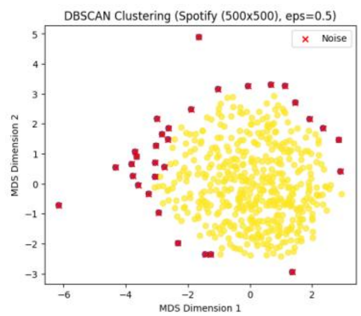
SPOTIFY DATASET: (USED SUBSET OF 500 X 500 FOR VAT, K MEANS, DBSCAN)

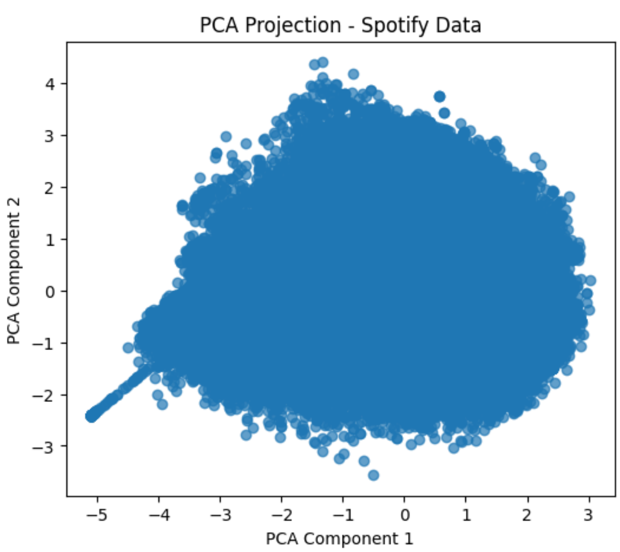


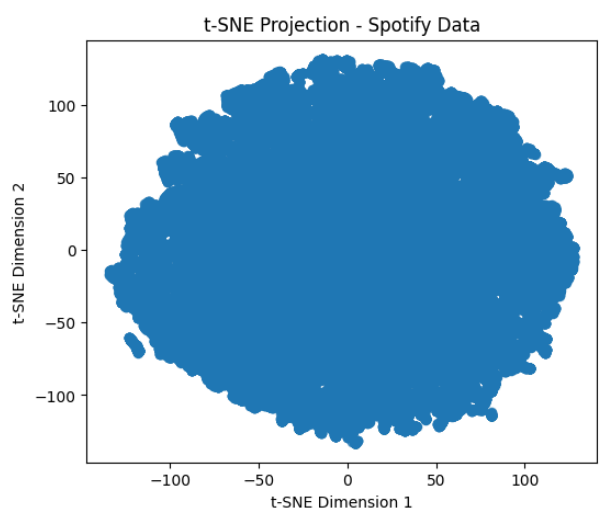




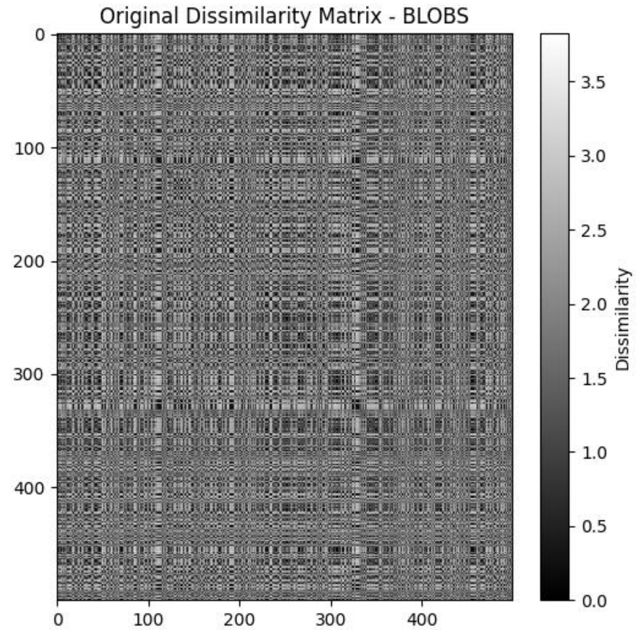


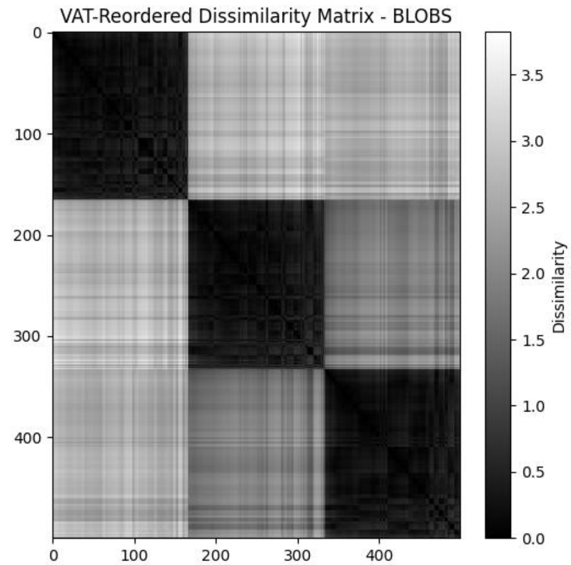


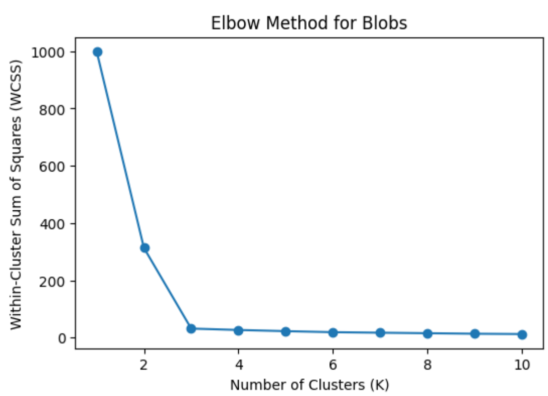


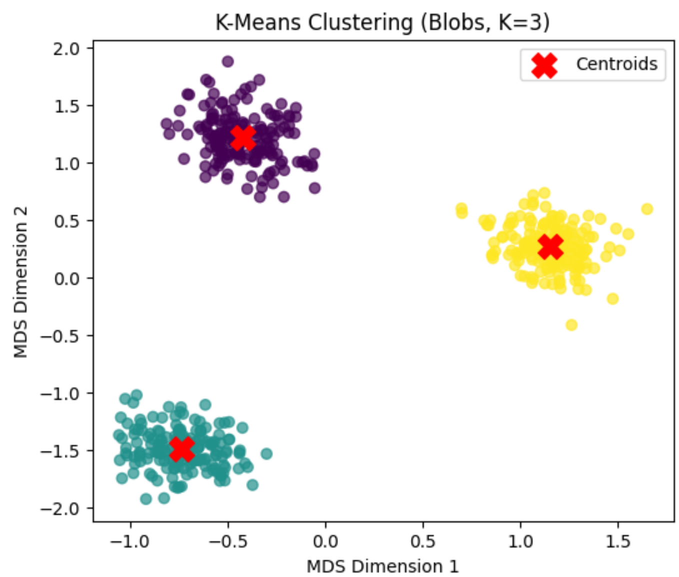


BLOBS DATASET:

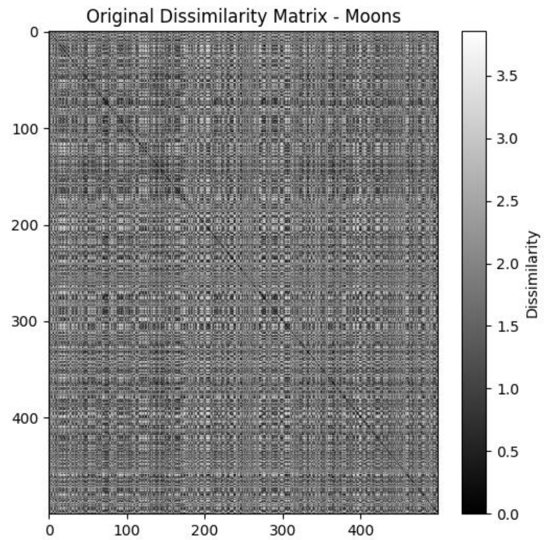


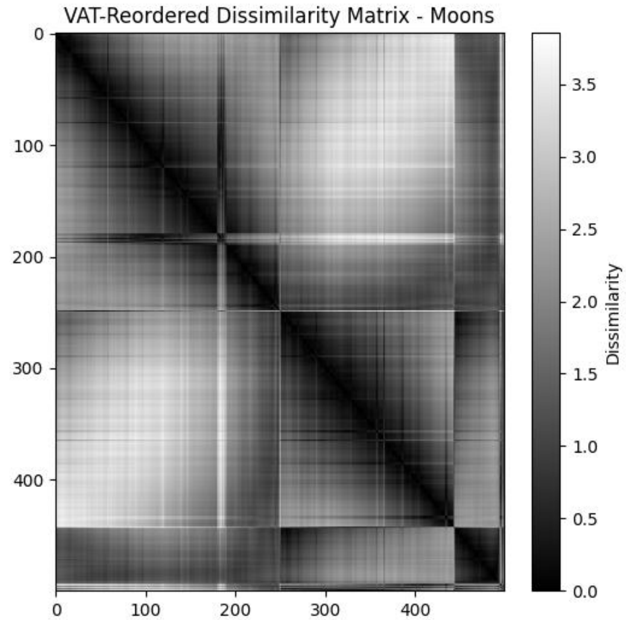


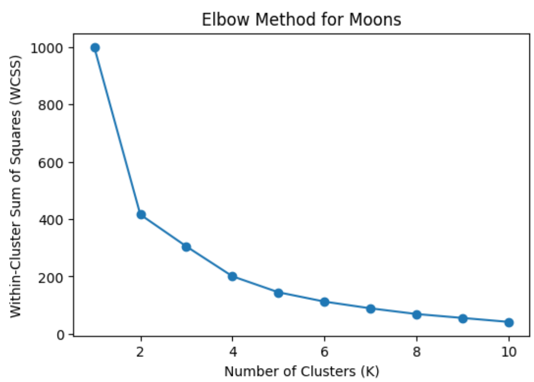


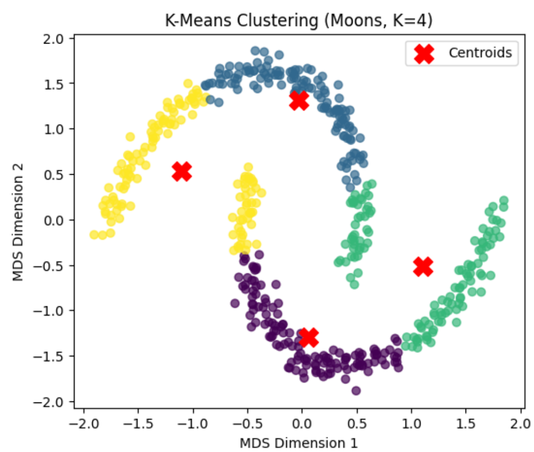


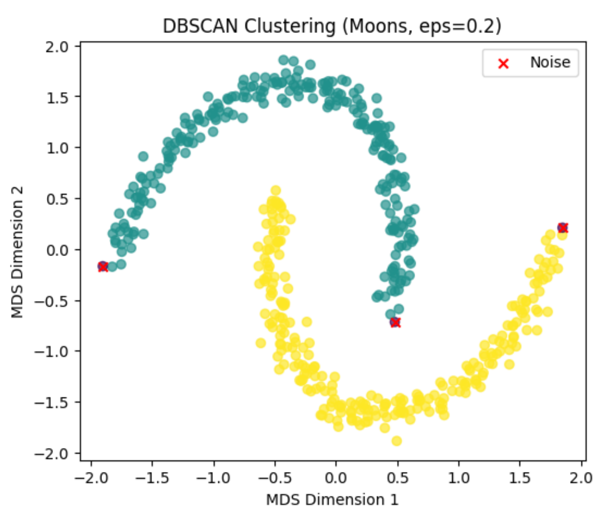
MOONS DATASET:



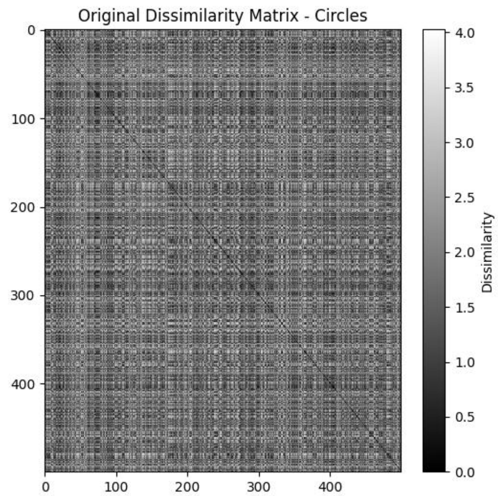


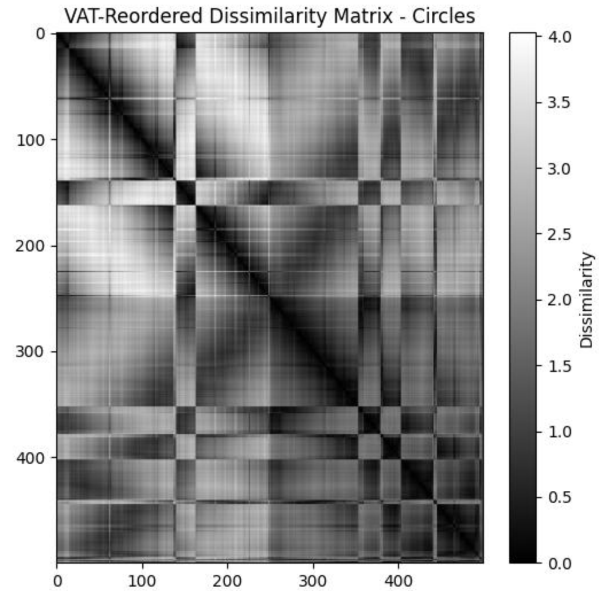


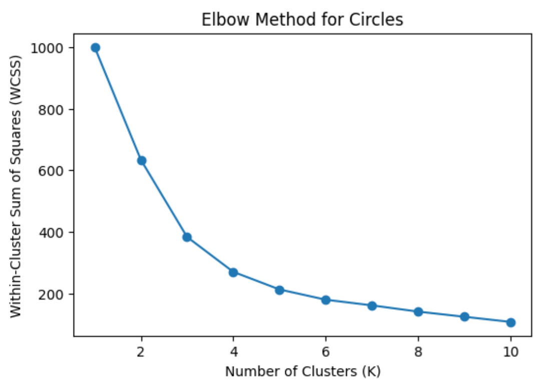


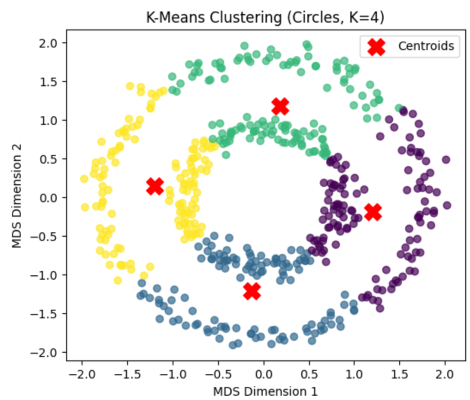


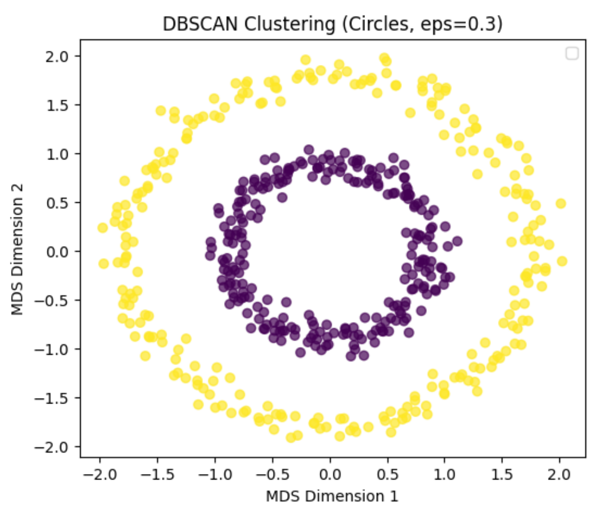
CIRCLES DATASET:



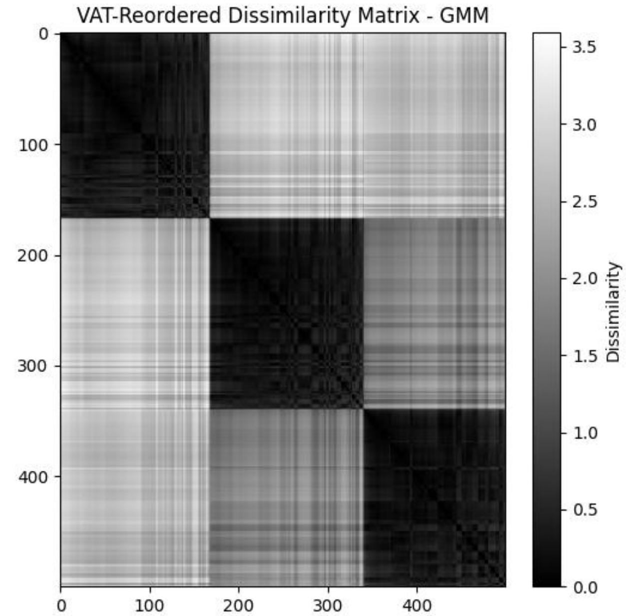


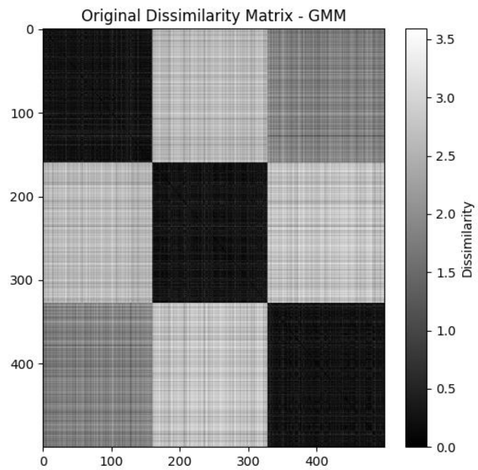


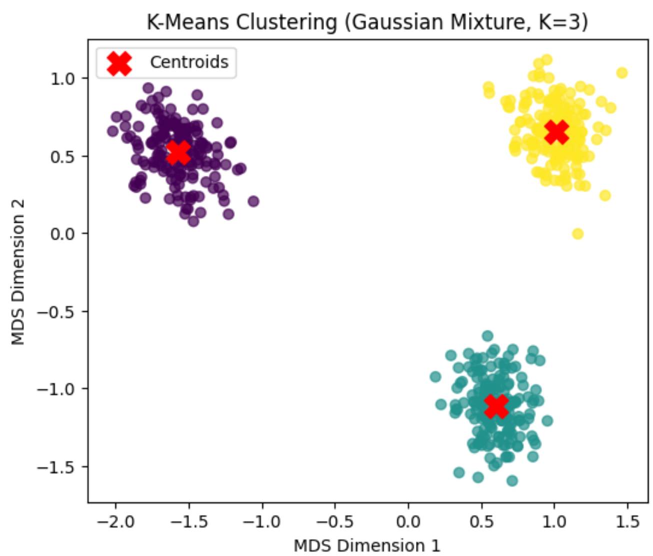


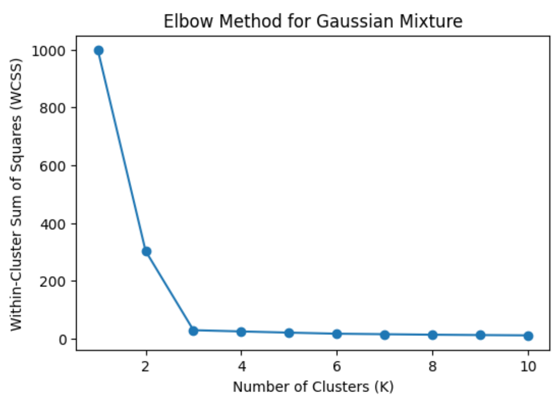


GMM DATASET:


VAT-Reordered Dissimilarity Matrix - GMM
0
100
200
300
400
0
100
200
300
400
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Dissimilarity


Original Dissimilarity Matrix - GMM
0
100
200
300
400
0
100
200
300
400
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Dissimilarity


K-Means Clustering (Gaussian Mixture, K=3)
Centroids
1.0
0.5
0.0
−0.5
−1.0
−1.5
MDS Dimension 2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
MDS Dimension 1


Elbow Method for Gaussian Mixture
1000
800
600
400
200
0
Within-Cluster Sum of Squares (WCSS)
2
4
6
8
10
Number of Clusters (K)