Project Title: VAT: A Tool for Visual Assessment of (Cluster) Tendency & it's Algorithm

Group: 4

Subject: DSA Foundations of Statistical Analysis and Machine Learning

Professor: Ismail Lachheb

Prepared

By

Prasanna Kumar Adabala	prasanna-kumar.adabala@epita.fr
Brahma Reddy Maddireddy	brahma-reddy.maddireddy@epita.fr
Raghuram Munagala	raghuram.munagala@epita.fr
Mouna Priya Pokuru	mouna-priya.pokuru@epita.fr
Bhargavi Akula	bhargavi.akula@epita.fr

Contents:

1. Introduction

The Visual Assessment of Tendency (VAT), introduced by J.C. Bezdek and R.J. Hathaway in their paper "VAT: A Tool for Visual Assessment of (Cluster) Tendency," is a significant technique used to visually assess the presence of clustering structures within a dataset. Unlike traditional clustering algorithms that partition data into clusters, VAT provides a preliminary check to determine whether clusters exist in the data at all. This is critical because applying clustering methods to data without natural clusters can lead to misleading results.

VAT operates by transforming the dataset into a dissimilarity matrix, where each element represents the dissimilarity (or distance) between pairs of data points. The core idea is to reorder this dissimilarity matrix using a specific algorithm, so that similar data points are grouped together. This reordered matrix is then visualized as an intensity image (often grayscale), where dark blocks along the diagonal indicate the presence of potential clusters. The technique works effectively with both object data (e.g., data points represented as vectors in a feature space) and relational data (e.g., pairwise dissimilarities).

By providing a simple visual representation of clustering tendencies, VAT helps practitioners to determine if further clustering analysis is warranted, making it a crucial step in exploratory data analysis.

2. Key Aspects of the VAT Paper

The paper "VAT: A Tool for Visual Assessment of (Cluster) Tendency" lays the foundation for understanding and applying VAT in cluster analysis. The key aspects discussed in the paper include:

- **Problem Addressed**: VAT provides a solution to the fundamental question of whether clusters exist in a dataset, which is critical before applying traditional clustering methods that always partition data into clusters.
- **Methodology**: VAT works by converting the input data into a dissimilarity matrix and then applying a reordering algorithm to this matrix to reveal potential clusters. The reordered matrix is displayed as an intensity image where clusters are indicated by dark blocks along the diagonal.
- **Data Representation**: VAT can be applied to two types of data representations:

    1. **Object Data**: This is the standard form where each data point is represented as a vector in a multi-dimensional feature space. The dissimilarity matrix is computed by evaluating the pairwise distances (or dissimilarities) between the data points.
    2. **Relational Data**: In this case, the dissimilarities between the data points are provided directly as a pairwise distance matrix, and VAT operates on this matrix without the need for explicit feature vectors.

- **Dissimilarity Matrix Calculation**: The VAT algorithm starts by calculating a **dissimilarity matrix** for the dataset. If the dataset is already in the form of dissimilarities (i.e., the pairwise distances between data points are provided), VAT uses that directly. If the data is in the form of feature vectors (object data), VAT first

computes the dissimilarities using a chosen metric, such as Euclidean distance, Manhattan distance, or another appropriate measure.

When the dataset provides **similarities** instead of dissimilarities, VAT converts these into dissimilarities. The formula used for this conversion is:

$$R_{ij} = S_{max} - S_{ij}$$

Where:

- R_{ij} is the dissimilarity between points i and j,
- S_{max} is the similarity between points i and j,
- S_{ij} is the maximum similarity in the matrix.

- **Reordering Algorithm**: The algorithm rearranges the data points to group similar points together, starting from the pair of points with the largest dissimilarity.
- **Visualization**: The reordered dissimilarity matrix is shown as a grayscale image, with darker regions along the diagonal suggesting potential clusters.

3. VAT algorithm Implementation Details:

The VAT algorithm was implemented in Python using SciPy, sci-kit learn, pandas, NumPy and matplotlib, encapsulating the key concepts described in the paper.

- The following sections describe the core methods used in the implementation.

3.1 __init__ (self, normalize=True, colormap='gray_r', n_samples_max=5000):

- Initializes the VAT class with options for normalizing the dissimilarity matrix, selecting the colormap, and setting the maximum sample size.
- This ensures the VAT algorithm can handle large datasets efficiently and visualize results in the chosen style.

```
class VAT:

    def __init__(self, normalize=True, colormap='gray_r', n_samples_max=5000):
        self.normalize = normalize
        self.n_samples_max = n_samples_max
        self.cmap = plt.cm.gray_r if colormap == 'gray_r' else LinearSegmentedColormap.from_list('vat_cmap', ['black', 'white'], N=256)
```

3.2 fit (self, data):

- This is the core method that ties together pre-processing, dissimilarity computation, VAT reordering, and storing the ordered dissimilarity matrix.
- This method processes the input data, calculates the dissimilarity matrix, and applies the VAT algorithm to reorder the matrix.

```python
def fit(self, data):
    if isinstance(data, pd.DataFrame):
        data = self.preprocess_data(data)
    if len(data) > self.n_samples_max:
        data = data[np.random.choice(len(data), self.n_samples_max, replace=False)]
    self.original_data = data.copy()
    # Compute dissimilarity matrix
    if data.shape[1] > 2 and self.is_nonlinear(data):
        self.R_ = self.geodesic_distance(data)
    else:
        self.R_ = squareform(pdist(data, 'euclidean'))
    if self.normalize:
        self.R_ = (self.R_ - self.R_.min()) / (self.R_.max() - self.R_.min())
    # VAT ordering
    self.order_ = self.vat_ordering(self.R_)
    self.R_ordered_ = self.R_[np.ix_(self.order_, self.order_)]
    return self
```

3.3 preprocess_data (self, data):

- This method ensures the dataset is clean and ready for dissimilarity matrix calculation. It handles missing data and converts categorical features into numerical values.
- It Pre-process the dataset by handling missing values and encoding categorical variables if the input is a Data Frame.

```python
def preprocess_data(self, data):
    num_cols = data.select_dtypes(include=[np.number]).columns
    cat_cols = data.select_dtypes(exclude=[np.number]).columns
    # Impute missing values
    data[num_cols] = SimpleImputer(strategy='mean').fit_transform(data[num_cols])
    data[cat_cols] = SimpleImputer(strategy='most_frequent').fit_transform(data[cat_cols])
    # Encode categorical variables
    if len(cat_cols) > 0:
        encoder = OneHotEncoder(drop='first', sparse_output=False)
        encoded = encoder.fit_transform(data[cat_cols])
        return np.hstack((data[num_cols].values, encoded))
    return data[num_cols].values
```

3.4 geodesic_distance(X):

- Computes the geodesic distance matrix between points using nearest neighbours and shortest path algorithms.
- Ensures accurate representation of complex distances in non-Euclidean space, which helps in identifying clusters.

```
def geodesic_distance(self, X):
    n_neighbors = min(15, X.shape[0]-1)
    nbrs = NearestNeighbors(n_neighbors=n_neighbors).fit(X)
    distances, indices = nbrs.kneighbors(X)

    dist_matrix = np.zeros((X.shape[0], X.shape[0]))
    for i in range(X.shape[0]):
        for j, d in zip(indices[i], distances[i]):
            if i != j:
                dist_matrix[i,j] = d
                dist_matrix[j,i] = d

    dist_matrix[dist_matrix == 0] = np.inf
    np.fill_diagonal(dist_matrix, 0)
    return shortest_path(dist_matrix, directed=False)
```

3.5 is_nonlinear (self, X):

- This method Performs PCA on the data and checks whether the first principal component explains less than 60% of the variance.
- Returns **True** if the data is deemed nonlinear based on this threshold, otherwise **False.**

```
    def is_nonlinear(self, X):
        pca = PCA(n_components=2).fit(X)
        return pca.explained_variance_ratio_[0] < 0.6
```

3.5 vat_ordering (self, R):

- Implements the VAT algorithm to reorder the dissimilarity matrix based on the minimum distances between points.
- Iteratively selects points starting from the maximum dissimilarity, grouping similar points together in the ordered matrix.

```
    def vat_ordering(self, R):
        n = R.shape[0]
        P = np.zeros(n, dtype=int)
        J = set(range(n))
        max_idx = np.unravel_index(np.argmax(R), R.shape)
        P[0] = max_idx[0]
        I = {P[0]}
        J.remove(P[0])
        for r in range(1, n):
            submatrix = R[np.array(list(I))[:, None], np.array(list(J))]
            min_val = np.min(submatrix)
            min_mask = submatrix == min_val
            candidates = np.argwhere(min_mask)
            if len(candidates) == 0:
                j_in_J = list(J)[0]
                P[r] = j_in_J
                I.add(j_in_J)
                J.remove(j_in_J)
                continue
            selected = candidates[np.argmax(candidates[:, 0])]
            i_in_I = list(I)[selected[0]]
            j_in_J = list(J)[selected[1]]

            P[r] = j_in_J
            I.add(j_in_J)
            J.remove(j_in_J)
        return P
```

3.5 plot_paper_style (self, title=None, figsize=(10, 5)):

- Plots both the original and VAT-ordered dissimilarity matrices side by side for visual comparison.
- Displays them with a grayscale colormap, providing a clear view of potential clusters as dark blocks along the diagonal.

```
def plot_paper_style(self, title=None, figsize=(10, 5)):
    """Create visualization matching paper format: random and ordered matrices side by side"""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)
    # Random dissimilarity matrix
    img1 = ax1.imshow(self.R_, cmap=self.cmap, aspect='equal',
                      vmin=0, vmax=1 if self.normalize else None)
    ax1.set_title('Random Dissimilarity Matrix', pad=15)
    ax1.axis('off')
    plt.colorbar(img1, ax=ax1, shrink=0.7)
    # Ordered VAT matrix
    img2 = ax2.imshow(self.R_ordered_, cmap=self.cmap, aspect='equal',
                      vmin=0, vmax=1 if self.normalize else None)
    ax2.set_title('Ordered Dissimilarity Matrix', pad=15)
    ax2.axis('off')
    plt.colorbar(img2, ax=ax2, shrink=0.7)
    if title:
        fig.suptitle(title, y=1.02)
    plt.tight_layout()
    return fig
```

4. Results and Test Cases

The following test cases showcase the application of the Visual Assessment of Tendency (VAT) algorithm on different datasets. These tests are designed to evaluate VAT's ability to identify clustering tendencies, assess its performance on both synthetic and real-world datasets, and examine how VAT handles datasets of varying complexities and sizes. The results will highlight VAT's effectiveness in visualizing potential clusters and structure in the data, providing a clear view of the underlying patterns before applying clustering techniques.

4.1 Test Case 1: Linear Data:

This test applies the VAT algorithm to synthetic linear data. The data consists of two variables with a linear relationship, and VAT is used to visualize the dissimilarity matrix. The goal is to check whether VAT can identify the lack of clustering structure in such data.

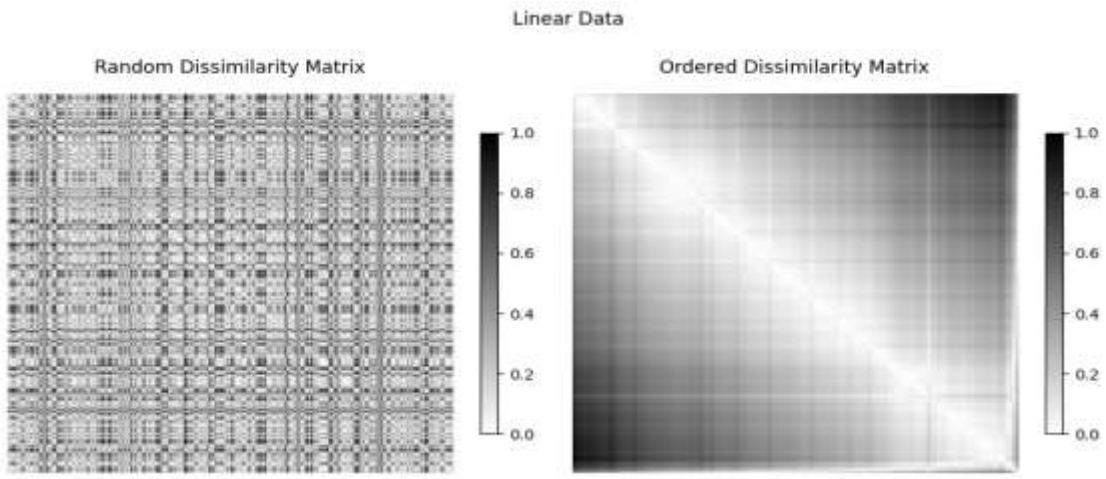

Fig. 4.1. Random and ordered dissimilarity images for Linear data.

4.2 Test Case 2: Circular Data:

This test uses circular data generated by make_circles, which has a clear clustering structure. The VAT algorithm helps assess if VAT can identify this structure, which would appear as two distinct clusters in the dissimilarity matrix.

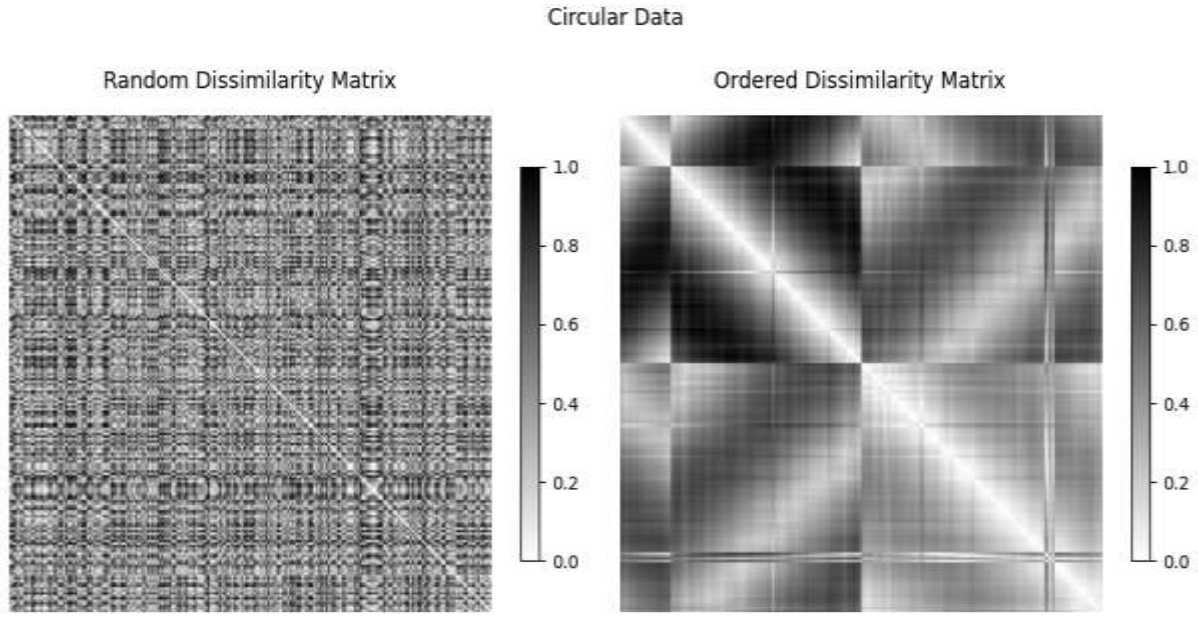

Fig. 4.2. Random and ordered dissimilarity images for circular data.

4.3 Test Case 3: IRIS Data:

The famous IRIS dataset is used here to test VAT on a well-known real-world dataset with three clusters. VAT is applied to the dissimilarity matrix to observe how well it detects the inherent clustering structure of the data.

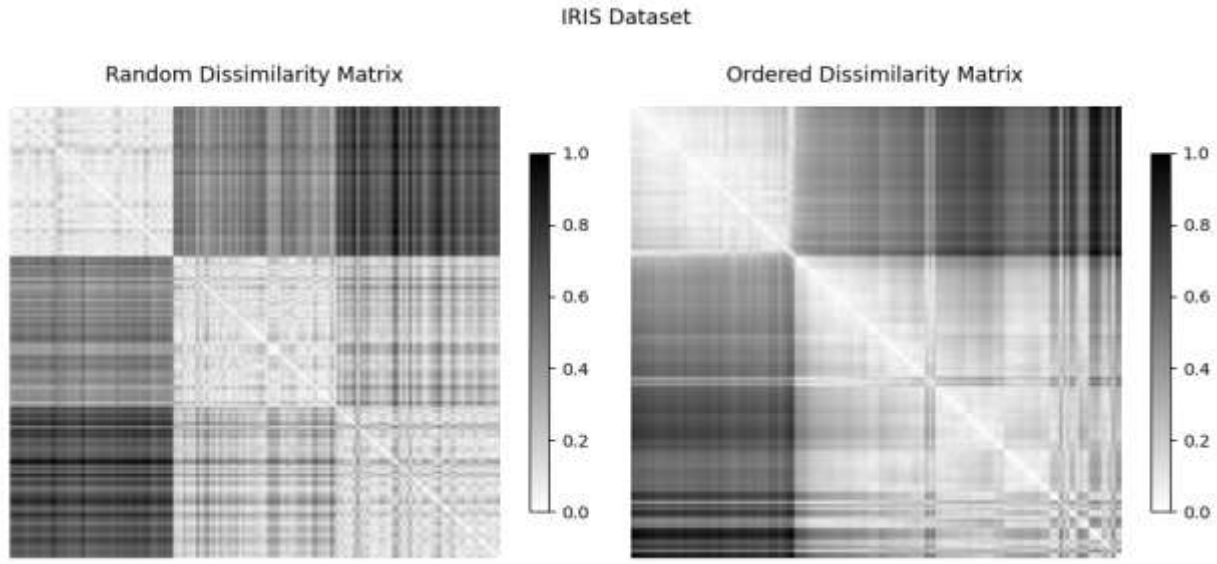

Fig. 4.3. Random and ordered dissimilarity images for IRIS data.

4.4 Test Case 4: Custom Dataset (Diabetes Prediction Dataset):

For this test, a real-world diabetes prediction dataset is used. VAT is applied to this dataset to identify any potential clusters in the dissimilarity matrix. This dataset contains multiple health-related features used to predict whether an individual has diabetes.

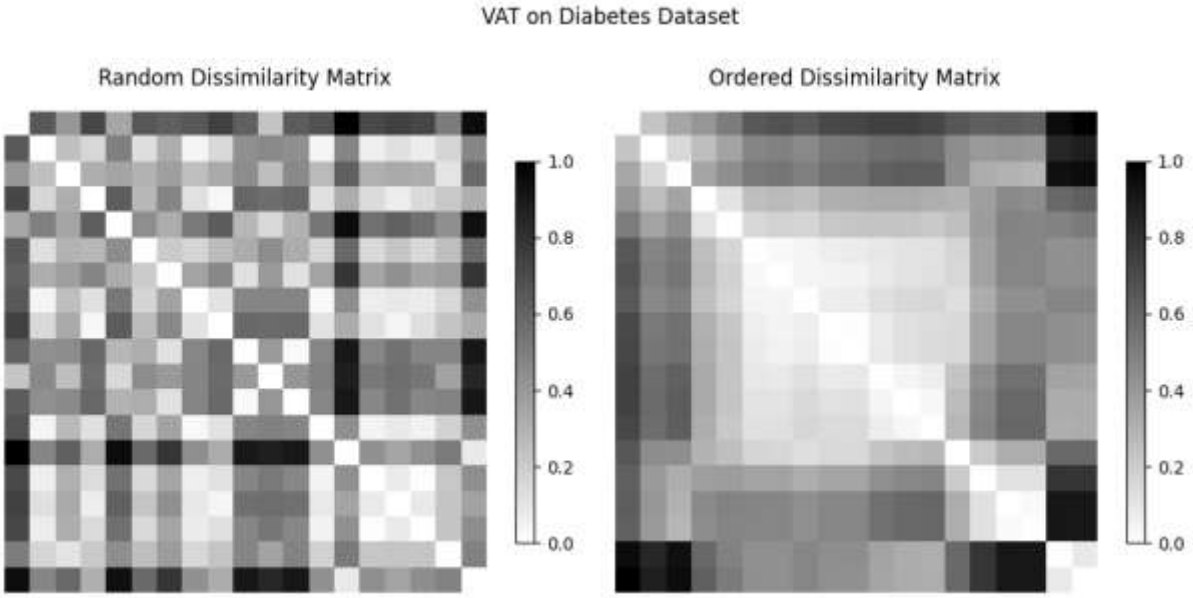

Fig. 4.4. Random and ordered dissimilarity images for Diabetes Prediction dataset.

4.5 Test Case 4: Large Random Dataset:

A random large dataset with 1000 samples and 5 features is used to evaluate the performance of VAT on larger datasets. This helps to assess VAT's scalability and whether it can still reveal useful patterns in more complex data.

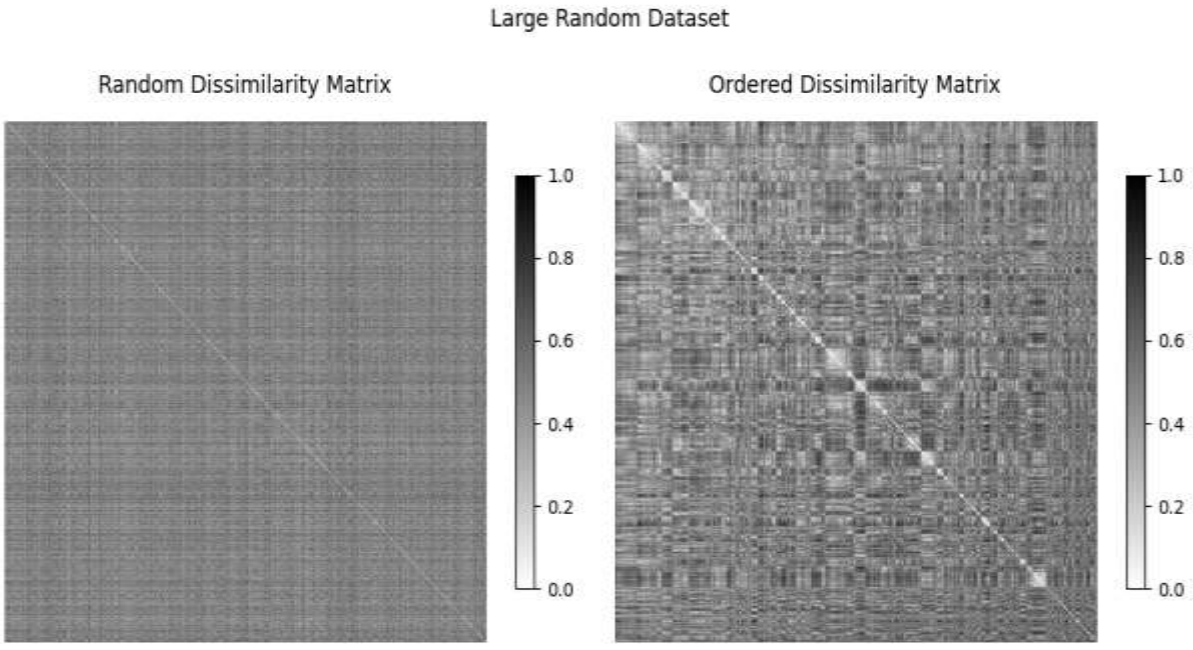

Fig. 4.5. Random and ordered dissimilarity images for Random data.

5. Learning Objectives from VAT:

1. **Visual Clustering Assessment**: VAT is a valuable tool for visually assessing the clustering tendency of a dataset before applying clustering algorithms. It helps identify whether natural groupings exist, saving time and resources.
2. **Data Pre-processing**: I learned the importance of pre-processing, such as handling missing values and converting categorical features, to ensure that VAT works effectively. This step is crucial for preparing the data for analysis.
3. **Dissimilarity Matrix**: VAT relies on the dissimilarity matrix, and choosing the right distance metric (e.g., Euclidean) is key to obtaining meaningful visualizations that reflect the underlying structure of the data.
4. **Reordering Algorithm**: VAT's reordering method efficiently groups similar data points, highlighting potential clusters. The reordered dissimilarity matrix shows dark diagonal blocks that represent groupings.
5. **Dataset Types**: VAT works well on both synthetic and real-world datasets. For synthetic datasets with clear clusters, VAT highlighted distinct groupings, while on real-world data like the diabetes dataset, VAT showed less obvious clustering, showcasing its sensitivity to data quality.
6. **Visualization and Insights**: VAT's intensity images provided an intuitive way to visualize clustering tendencies. The ability to compare original and reordered matrices helped me spot potential clusters quickly.
7. **Scalability**: I learned that VAT can be computationally expensive, especially for large datasets. Subsampling techniques help improve efficiency without sacrificing too much accuracy.
8. **VAT and Clustering Algorithms**: VAT complements traditional clustering algorithms by providing a visual check of clustering tendencies. It helps choose the right clustering method based on the data structure.
9. **Limitations**: VAT is not scalable for very large datasets, and it is a purely visual tool without direct cluster labels. It should be used in conjunction with other methods for more comprehensive analysis.
10. **Broader Learning**: This project reinforced the importance of data visualization in data science, particularly in exploratory analysis. VAT helped me understand how to assess data quality, detect patterns, and inform the choice of clustering methods.

6. Conclusion

The VAT algorithm proved to be a valuable tool for visually assessing the clustering tendency of datasets, helping to identify inherent structures before applying clustering algorithms. Through various test cases on synthetic and real-world datasets, I gained insights into the importance of data pre-processing, dissimilarity matrix computation, and the impact of reordering on cluster visualization. While VAT effectively highlights potential clusters, it has scalability limitations for very large datasets, necessitating techniques like subsampling. Despite these constraints, VAT serves as a useful exploratory tool in data science, complementing traditional clustering methods and reinforcing the importance of visualization in understanding data distributions and structures.

7. References

- VAT Algorithm Paper
- Python Libraries: NumPy, Pandas, Sci-kit learn SciPy and Matplotlib.