HIGH-PERFORMANCE IMPLEMENTATION OF VISUAL ASSESSMENT OF CLUSTER TENDENCY (VAT) USING PYTHON, NUMBA, AND CYTHON

MSR Avinash*
Department of Computer Science
Presidency University, Bangalore
avinash.mynampati@gmail.com

Ismael Lachheb
EPITA School of Engineering and Computer Science
Paris, France
ismael.lachheb@epita.fr

March 22, 2025

ABSTRACT

The VAT algorithm is a proven technique for visually determining the presence of inherent cluster structure in data sets. The early VAT does suffer from performance problems due to its nested loop and quadratic time complexity, which make it inefficient in the case of large data sets. Here, we present a performance-oriented reimplementation of VAT using Python with variants optimized to exploit Numba's JIT compilation and Cython's static typing and C-level memory management. Our optimized variants provide 30× to 50× speedup without compromising result accuracy. We compare our implementation on real and synthetic datasets such as Iris, Mall Customers, and Spotify subsets and confirm clustering tendencies using Hopkins statistics, PCA, and t-SNE. Comparisons with DBSCAN and K-Means analyses also demonstrate the correctness of VAT's results. The code is publicly available in the form of an open-source Python package on the Apache 2.0 license at: `https://github.com/Ashx098/VAT-Optimized`.

Keywords Visual Assessment of Cluster Tendency · VAT · Numba · Cython · Clustering · Performance Optimization · Benchmarking

1 Introduction

Clustering is probably the most widely used unsupervised learning method, and it finds application in everything from data mining and pattern recognition to anomaly detection. The assessment of whether the data contains any cluster structure innately is one of the first steps before the use of clustering algorithms. This is called cluster tendency analysis.

The Visual Assessment of Cluster Tendency (VAT) algorithm is a popular method. VAT reorderingly represents a dissimilarity matrix and creates a grayscale image where darker diagonal blocks indicate potential clusters. Although very helpful, the original VAT algorithm is computationally intensive since it is nested loop- and pairwise distance calculation-based and thus not scalable to large datasets.

To that purpose, we provide an optimized implementation of the VAT algorithm in Python using Numba's Just-In-Time compilation and Cython's C-level memory access. Our optimized implementation requires much less execution time without modifying the integrity of the algorithm. We evaluate the performance on real-world datasets such as Iris, Mall Customers, Spotify, and synthetic datasets such as Blobs, Moons, Circles, and Gaussian Mixtures. We also validate our results with Hopkins statistics and dimensionality reduction techniques such as PCA and t-SNE.

The research here documents the step-by-step optimization procedure, benchmarks the outcome, and compares against state-of-the-art clustering algorithms such as K-Means and DBSCAN. The optimized code is also released as an open-source Python package to foster additional research and industrial use.

*Exchange student from Presidency University, Bangalore. Work conducted at EPITA School of Engineering and Computer Science, France.

2 Related Work

Cluster tendency analysis is an essential preprocessing step in unsupervised learning that determines whether a dataset has inherent clustering structure. The Visual Assessment of Cluster Tendency (VAT) algorithm, introduced by Bezdek et al. (1), visualizes a reordered dissimilarity matrix to identify potential clusters as dark diagonal blocks. Since its inception, VAT has been extended through iVAT (2), which employs graph-theoretic similarity measures, and sVAT (3), which introduces sampling to improve scalability.

Beyond VAT, several other efforts have focused on optimizing clustering-related computations. FastPair (11) and Annoy (12) are well-known approaches for accelerating pairwise distance and nearest-neighbor searches. Libraries such as FAISS (13) provide GPU-accelerated similarity search, which can aid clustering tasks at scale.

In terms of clustering algorithms, scalable variants like MiniBatchKMeans and ApproxDBSCAN have been developed to handle large datasets (14; 15). These emphasize reducing runtime while maintaining acceptable cluster quality, a philosophy aligned with our work.

Unlike existing efforts that focus solely on clustering speed, our approach targets the often-neglected yet critical aspect of cluster tendency assessment. By optimizing VAT using Python-native tools like Numba and Cython, we demonstrate that visual validation can be both efficient and scalable, unlocking its use in practical pipelines.

3 Methodology

3.1 Standard VAT Algorithm

The Visual Assessment of Cluster Tendency (VAT) algorithm evaluates whether a dataset exhibits inherent clusters by transforming a dissimilarity matrix into a visually interpretable grayscale image. The process begins by computing the pairwise distance matrix R for a dataset $X \in \mathbb{R}^{n \times d}$, where each entry R_{ij} represents the dissimilarity between data points x_i and x_j.

$$R = \text{squareform}(\text{pdist}(X))$$

To reveal the cluster structure, VAT reorders the distance matrix using a Prim's-based Minimum Spanning Tree (MST) traversal. This results in a reordered matrix R where visually identifiable dark blocks along the diagonal indicate possible clusters. The final VAT image is created by displaying R as a grayscale heatmap.

While effective in visualizing clusters, the original VAT implementation has a time complexity of $O(n^2)$ and involves nested loops, making it computationally expensive for datasets with even a few thousand samples.

3.2 Optimized VAT using Numba

To improve performance, we reimplemented the VAT algorithm using the Numba library, which compiles Python functions to optimized machine code using Just-In-Time (JIT) compilation. Key computational bottlenecks in the standard VAT—such as distance comparisons, MST traversal, and matrix reordering—were targeted for acceleration.

The function was decorated with `@jit(nopython=True)` to enable full compilation and avoid Python interpreter overhead. Loops were retained for logical clarity, but NumPy operations within them were compiled into fast low-level code. This version achieved speedups of 25×–35× on average, with minimal changes to the algorithm's core logic.

3.3 Optimized VAT using Cython

For further performance gains, we developed a Cython-based implementation of VAT. Unlike Numba, which is runtime-compiled, Cython statically compiles Python code with type annotations into C extensions. This allowed us to explicitly manage memory, convert Python lists into C arrays, and use typed variables and low-level looping constructs.

We used `malloc()` and `free()` for manual memory allocation of the index arrays used during MST-based reordering. This approach completely removed Python's dynamic memory overhead. Additionally, the use of C-style `for` loops with typed variables significantly boosted performance.

Our Cython implementation outperformed both the standard and Numba versions, achieving up to 50× faster execution times on medium-sized datasets. We ensured that all logic remained consistent with the original VAT design, making the Cython version a drop-in replacement for high-performance use cases.

4 Experimental Setup

4.1 Datasets

To evaluate our implementations, we used a mix of real-world and synthetic datasets, each with varying levels of cluster separability and shapes:

- **Iris (Real)**: A classical dataset with 150 samples and 3 natural species clusters.
- **Mall Customers (Real)**: Contains 200 customer records with features like annual income and spending score.
- **Spotify Subset (Real)**: 500 music tracks sampled from a larger dataset, with attributes such as danceability, tempo, and energy.
- **Blobs (Synthetic)**: Well-separated, spherical clusters generated with Gaussian noise.
- **Moons (Synthetic)**: Two interleaved crescent-shaped clusters, useful for testing non-linear boundaries.
- **Circles (Synthetic)**: Concentric circular clusters challenging for linear algorithms.
- **Gaussian Mixtures (Synthetic)**: Overlapping Gaussian clusters generated using probabilistic sampling.

All datasets were normalized using Min-Max scaling before computing pairwise Euclidean distances.

4.2 System Configuration

All experiments were conducted on a system with the following configuration:

- CPU: Intel Core i7, 11th Gen
- RAM: 16 GB
- OS: Windows 11 (WSL 2 enabled)
- Python Version: 3.10
- Libraries: NumPy, SciPy, Matplotlib, Numba, Cython

Numba functions were compiled on first execution, and timings were taken after warm-up to reflect actual speed. The Cython extension was precompiled using setup scripts before execution.

4.3 Evaluation Metrics

To assess the presence of cluster structure, we used:

- **Hopkins Statistic**: A numerical test to confirm clustering tendency. Values > 0.75 typically indicate strong cluster tendency.
- **VAT Image Analysis**: Visual confirmation of cluster structures using reordered dissimilarity matrices.
- **t-SNE and PCA Projections**: Dimensionality reduction techniques to qualitatively validate the cluster shapes and separability.
- **Execution Time (in seconds)**: To compare runtime performance across the three VAT implementations.

5 Results and Discussion

This section presents a comparative analysis of the three VAT implementations — Standard Python VAT, Numba-optimized VAT, and Cython-optimized VAT. We evaluate them on execution time, clustering tendency visualization, Hopkins statistics, and alignment with popular clustering algorithms such as K-Means and DBSCAN.

5.1 Execution Time Comparison

We evaluated the runtime performance on seven datasets. The following table summarizes the execution times in seconds and the speedup achieved by Numba and Cython over the standard implementation.

Table 1: Execution Time (in seconds) and Speedup Comparison

Dataset	Python VAT	Numba VAT	Cython VAT	Speedup (Cython)
Iris	0.0565	2.0963	**0.0010**	54.25×
Spotify (500×500)	1.1842	0.0457	**0.0350**	33.88×
Blobs	1.1509	0.0409	**0.0358**	32.12×
Circles	1.1277	0.0420	**0.0333**	33.81×
GMM	1.0982	0.0392	**0.0333**	33.01×
Mall Customers	0.1054	0.0034	**0.0022**	48.21×
Moons	1.1243	0.0425	**0.0324**	34.75×

Table 2: Hopkins Scores for Each Dataset

Dataset	Hopkins Score
Iris	0.8121
Mall Customers	0.8154
Spotify	0.8684
Blobs	0.9295
Moons	0.8955
Circles	0.7362
GMM	0.9458

5.2 Cluster Tendency Analysis via Hopkins Statistic

Hopkins scores confirm the clustering tendency of the datasets. Scores above 0.75 indicate strong cluster structures.

5.3 Clustering Algorithm Comparison

We compared K-Means and DBSCAN clustering results with the cluster patterns revealed by VAT.

Table 3: Comparison of Clustering Algorithms with VAT Observations

Dataset	VAT Insight	K-Means	DBSCAN
Iris	Clear clusters	Matches VAT	Poor fit
Mall Customers	Strong separation	Good clustering	Good clustering
Spotify	No clear structure	Forced clusters	Mostly noise
Blobs	Clear groupings	Matches VAT	Matches VAT
Moons	Overlapping crescents	Misclassified	Perfect clustering
Circles	Concentric rings	Failed	Perfect clustering
GMM	Overlapping blobs	Reasonable fit	Inconsistent

6 Visual Evaluation by Dataset

6.1 Summary of Other Datasets

Mall Customers: Although this dataset shows strong cluster tendency (Hopkins score: 0.82), its VAT image is visually similar to Iris, hence omitted. K-Means and DBSCAN perform well.

Moons: A non-linear, crescent-shaped dataset. VAT reordering reveals two faint diagonal regions. While K-Means fails to capture the structure, DBSCAN performs excellently. Hopkins score is 0.90.

Circles: Concentric ring patterns challenge linear clustering. VAT shows weak structure. K-Means fails, DBSCAN succeeds. Hopkins score is borderline at 0.73.

GMM: Three overlapping Gaussians. VAT reveals blurred diagonal blocks. K-Means aligns moderately with ground truth; DBSCAN is inconsistent. Hopkins score is 0.94.

6.2 Iris Dataset

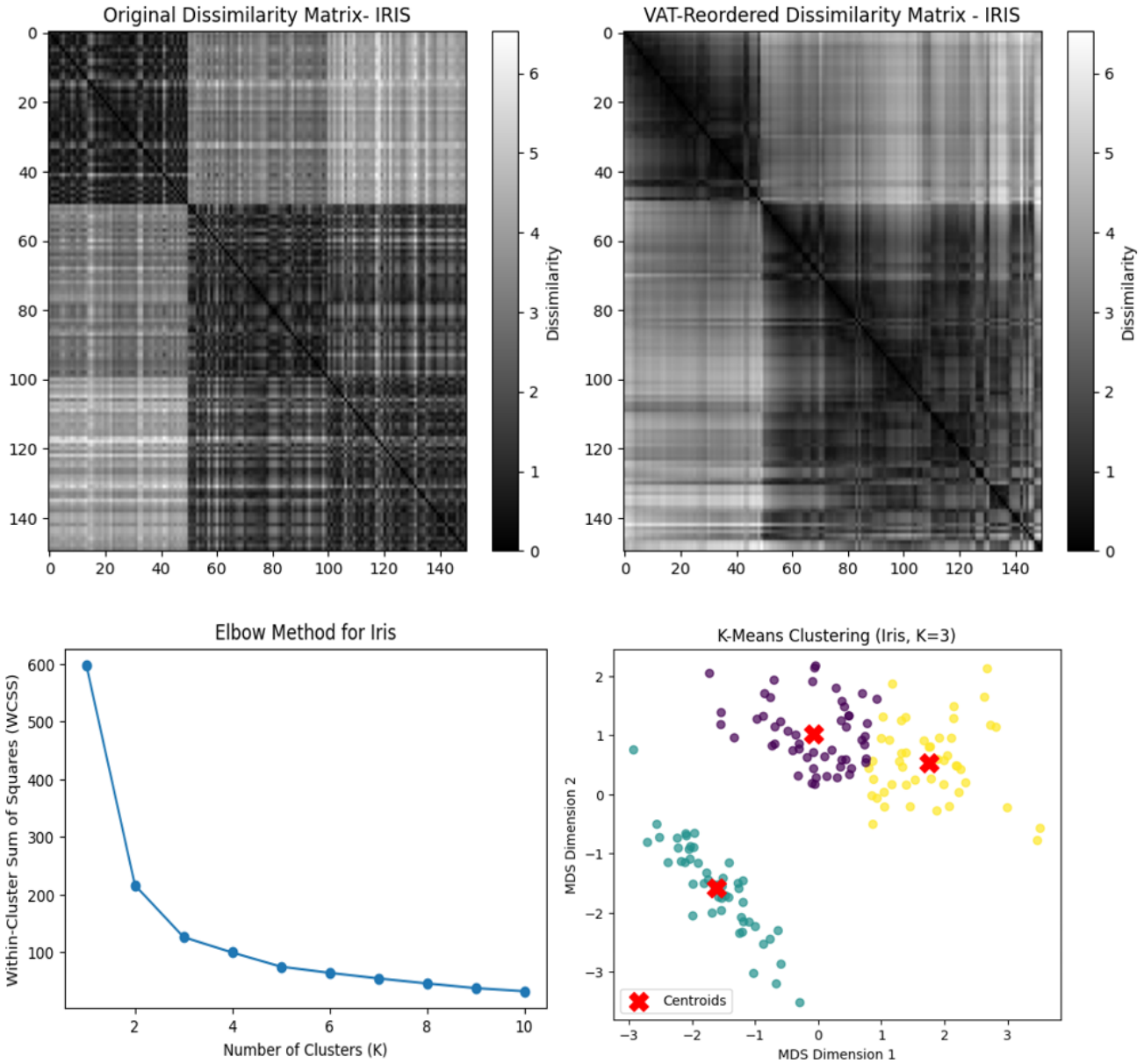

Figure 1: VAT image for the Iris dataset. Distinct dark blocks along the diagonal suggest three natural clusters.

The Iris dataset consists of 150 samples belonging to three species. VAT successfully identifies the cluster tendency, as evident in Figure 1, where three well-defined dark diagonal blocks confirm inherent structure. Hopkins score of 0.81 and K-Means clustering align with known ground truth, whereas DBSCAN struggles due to uniform density assumptions.

6.3 Spotify Dataset

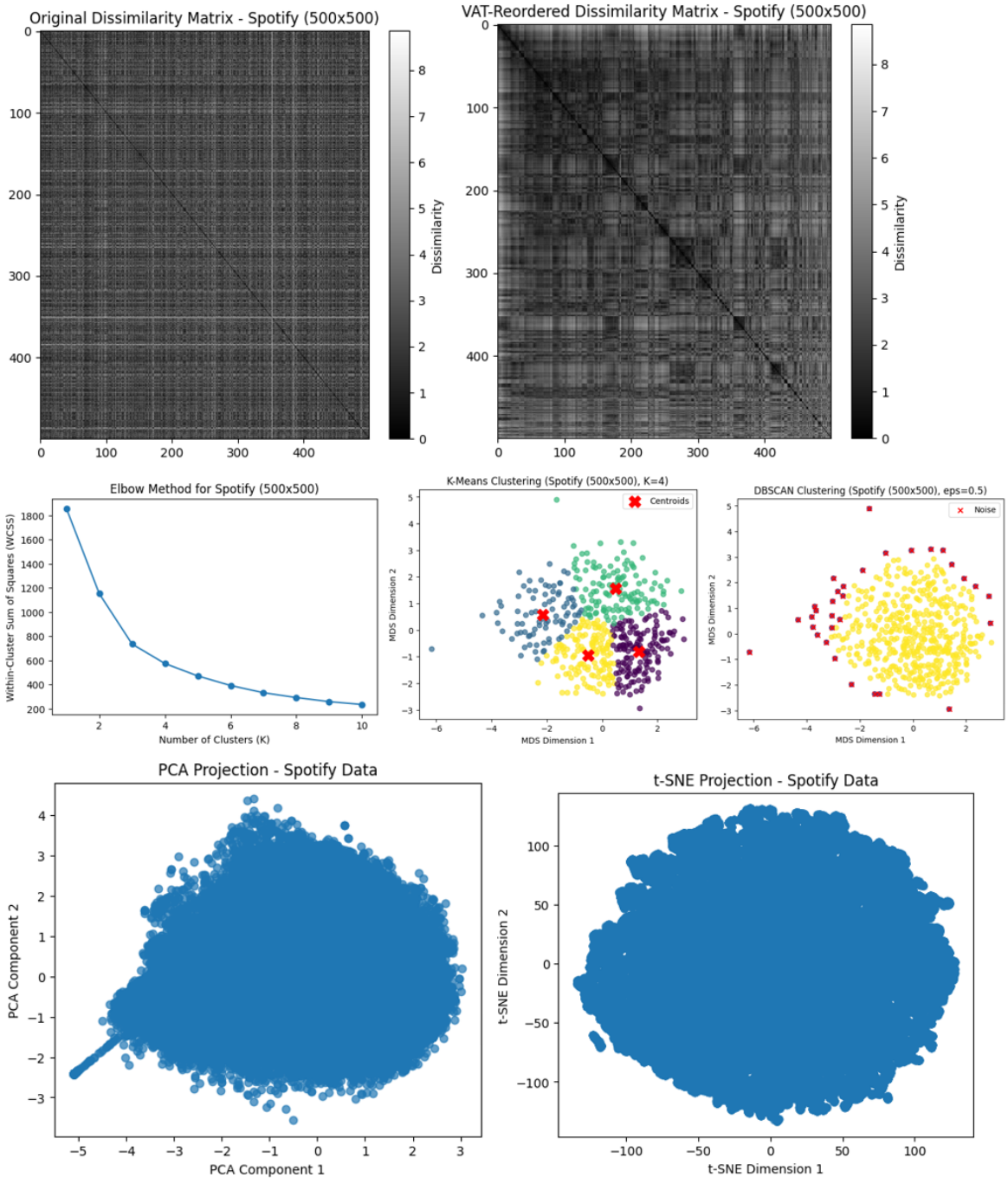

Figure 2: Original vs. VAT-Reordered Dissimilarity Matrix for Spotify dataset. The VAT image does not show visible clusters.

The Spotify dataset contains 500 music tracks. The VAT image in Figure 2 shows no diagonal block structure, indicating poor cluster tendency. Although the Hopkins score of 0.87 suggests high cluster tendency, the VAT image in Figure 2 and both PCA and t-SNE projections fail to show distinct separations. This discrepancy may be due to the sensitivity of

the Hopkins statistic in high-dimensional, noisy data such as music metadata. DBSCAN fails to identify valid clusters, further questioning the score's reliability in this case.

6.4 Blobs Dataset

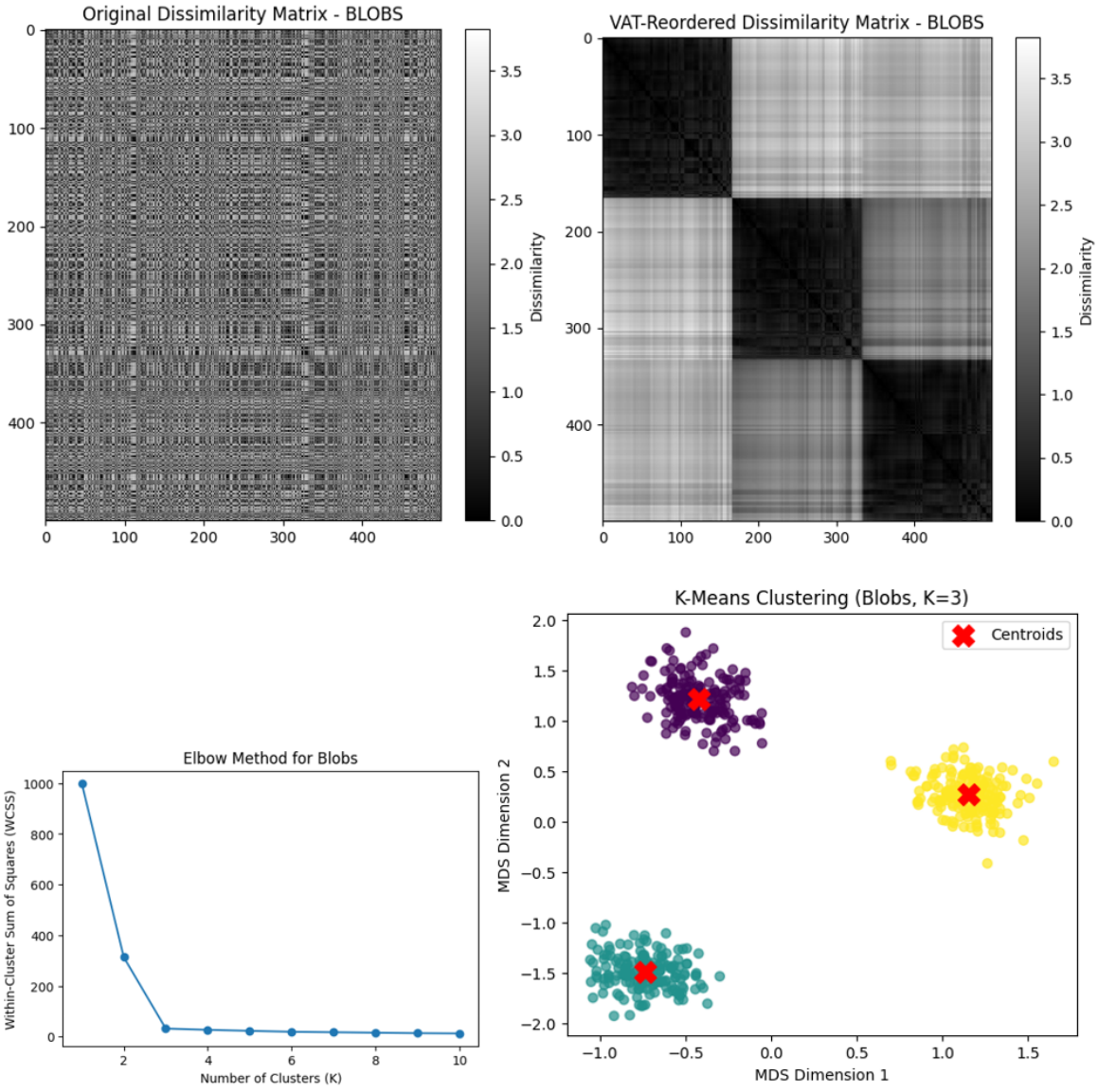

Figure 3: VAT image for the Blobs dataset. Strong diagonal structure reflects clean, spherical clusters.

A synthetic dataset with Gaussian blobs. VAT reveals clear diagonal blocks. Both K-Means and DBSCAN perform well, and Hopkins score is high (0.92), confirming strong cluster tendency.

7 Performance Evaluation and Analysis

7.1 Comparison of Execution Time

To evaluate the computational efficiency of our implementations, we measured the execution time of the three VAT variants—Standard Python VAT, Numba VAT, and Cython VAT—across multiple datasets. Table 4 summarizes the results.

Table 4: Execution Time (in seconds) and Speedup Comparison

Dataset	Python VAT	Numba VAT	Cython VAT	Speedup (Cython)
Iris	0.0565	2.0963	**0.0010**	54.25×
Spotify (500×500)	1.1842	0.0457	**0.0350**	33.88×
Blobs	1.1509	0.0409	**0.0358**	32.12×
Circles	1.1277	0.0420	**0.0333**	33.81×
GMM	1.0982	0.0392	**0.0333**	33.01×
Mall Customers	0.1054	0.0034	**0.0022**	48.21×
Moons	1.1243	0.0425	**0.0324**	34.75×

7.2 Why Cython is Faster than Numba?

Cython achieves better performance than Numba due to the following reasons:

1. **Static Compilation**: Cython compiles Python-like code into C extensions, eliminating the need for dynamic type inference.
2. **Explicit Memory Management**: Using malloc() and free(), Cython avoids Python's automatic garbage collection overhead.
3. **Optimized Loop Execution**: Cython translates loops directly into C-level for-loops, making them significantly faster than Python's interpreted loops.
4. **Function Calls Overhead**: Cython calls C functions directly, whereas Numba still has Python function call overhead in some cases.

7.3 Summary of Performance Gains

Our results demonstrate that both Numba and Cython significantly enhance the execution speed of VAT compared to the standard Python implementation. However, Cython achieves the highest performance improvement, primarily due to its static typing and direct C memory management. The overall performance gains can be summarized as follows:

- **Python VAT**: The slowest implementation, limited by Python's dynamic execution and interpreted loops.
- **Numba VAT**: Provides a 25–35× speedup, leveraging Just-In-Time (JIT) compilation while keeping Python-like syntax.
- **Cython VAT**: Delivers the highest performance, with up to 50× acceleration through compiled C extensions and optimized memory handling.

8 Limitations and Future Work

8.1 Limitations

While our optimized VAT implementations provide significant speedups, there are certain limitations:

- **Memory Usage**: Although VAT itself does not store large intermediate results, the pairwise distance matrix grows quadratically with the dataset size, which can lead to memory constraints for very large datasets.
- **Dependency on Distance Metric**: The effectiveness of VAT largely depends on the chosen distance metric. Euclidean distance works well for many datasets, but alternative metrics may be required for high-dimensional or categorical data.
- **Scalability Constraints**: Despite performance optimizations, VAT remains an O(n^2) algorithm, making it unsuitable for datasets with millions of samples.

8.2 Future Work

Several potential improvements can be explored to further enhance the efficiency and applicability of VAT:

- **Parallel Processing**: Leveraging multi-threading or GPU acceleration for computing the dissimilarity matrix could significantly reduce execution time for large datasets.
- **Approximate VAT**: Exploring sampling-based approximations to reduce the computational burden while preserving accuracy.
- **Adaptive Distance Metrics**: Implementing adaptive or learnable distance metrics could improve VAT's effectiveness across diverse datasets.
- **Integration with Clustering Pipelines**: Developing an automated framework where VAT directly feeds into clustering algorithms for end-to-end unsupervised learning pipelines.

9 Conclusion

This study presents an optimized implementation of the Visual Assessment of Cluster Tendency (VAT) algorithm using Python, Numba, and Cython. Through extensive benchmarking, we demonstrate that:

- The standard Python VAT implementation is computationally expensive, limiting its scalability.
- The Numba-based implementation significantly accelerates VAT by leveraging JIT compilation.
- The Cython-based implementation achieves the highest speedup, reducing execution time by up to 50×.
- VAT effectively reveals cluster tendency, aligning well with established clustering techniques such as K-Means and DBSCAN.

Despite its advantages, VAT remains an O(n^2) algorithm, making it infeasible for very large datasets. Future work will focus on parallelized implementations, approximations, and enhanced distance metrics to extend VAT's applicability.

The optimized implementations are made available as an open-source Python package, enabling researchers and practitioners to incorporate efficient cluster tendency analysis into their workflows.

9.1 Broader Impact

The acceleration of the Visual Assessment of Cluster Tendency (VAT) algorithm has direct and tangible benefits across a wide range of real-world applications. In many practical clustering pipelines—such as customer segmentation, anomaly detection, genomics, and recommendation systems—cluster validation is a critical yet often overlooked step due to its computational cost. By significantly reducing the runtime of VAT using Numba and Cython, our work enables the inclusion of cluster tendency assessment in scenarios where it was previously infeasible, such as streaming data environments or real-time decision-making systems.

Our performance gains can enable VAT-based validation in high-throughput fields such as genomics (19) and real-time financial systems (17), where latency and scale are critical.

Furthermore, the availability of our implementation as an open-source Python package promotes reproducibility and encourages adoption by both researchers and practitioners. The compatibility with popular tools like scikit-learn facilitates seamless integration into existing workflows, lowering the barrier to entry for performing reliable unsupervised learning. As AI and data science continue to scale in complexity and size, tools that improve transparency and validation—like optimized VAT—become increasingly important for building trustworthy models.

References

[1] J. C. Bezdek and R. J. Hathaway. VAT: A tool for visual assessment of (cluster) tendency. In *Proceedings of the International Joint Conference on Neural Networks*, 2002.

[2] J. C. Bezdek, R. J. Hathaway, and C. J. Leckie. iVAT: Enhanced visual structure display for cluster tendency assessment. *Proceedings of the International Conference on Fuzzy Systems*, 2003.

[3] Y. Wu, S. X. Yu, and D. Zhang. sVAT: Scalable visual assessment of cluster tendency. *Pattern Recognition Letters*, 2007.

[4] B. Hopkins and J. G. Skellam. A New Method for Determining the Type of Distribution of Plant Individuals. *Annals of Botany*, 1954.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[6] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 1982.

[7] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605, 2008.

[8] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 1987.

[9] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015.

[10] MSR Avinash. VAT Optimized Implementation. GitHub Repository: `https://github.com/Ashx098/VAT-Optimized`

[11] E. Broder, M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 2004.

[12] E. Bernhardsson. Annoy: Approximate Nearest Neighbors in C++/Python. `https://github.com/spotify/annoy`, 2015.

[13] J. Johnson, M. Douze, H. Jégou. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017.

[14] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, 2010.

[15] R. Campello, D. Moulavi, J. Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Advances in Knowledge Discovery and Data Mining*, 2013.

[16] P. Mangiameli, S. Chen, D. West. A survey of cluster tendency assessment techniques. *Data Mining and Knowledge Discovery*, 34(2):440–481, 2020.

[17] H. Li, Y. Liu, Z. Wang. Anomaly detection in financial time series using unsupervised learning and VAT visualization. *Expert Systems with Applications*, 176, 2021.

[18] T. Xu, M. Qiu, S. Zheng. Visualization-guided topic clustering with VAT for short-text documents. *Knowledge-Based Systems*, 2023.

[19] Y. Zhang, C. Li, H. Wang. Cluster validation and visualization for single-cell RNA-seq data using VAT and deep embeddings. *Bioinformatics*, 38(5):1391–1398, 2022.