BDA2025_FINAL report

Task Description

BDA2025 Final Project focuses on the usage of clustering method. The task is to analyze the relationships within this dataset and **classify the data into 4n – 1 clusters**, where n is the number of dimensions of the data.

The results will finally be evaluated based on the Fowlkes–Mallows Index (FMI), which measures the similarity between your clustering results and a hidden ground truth.

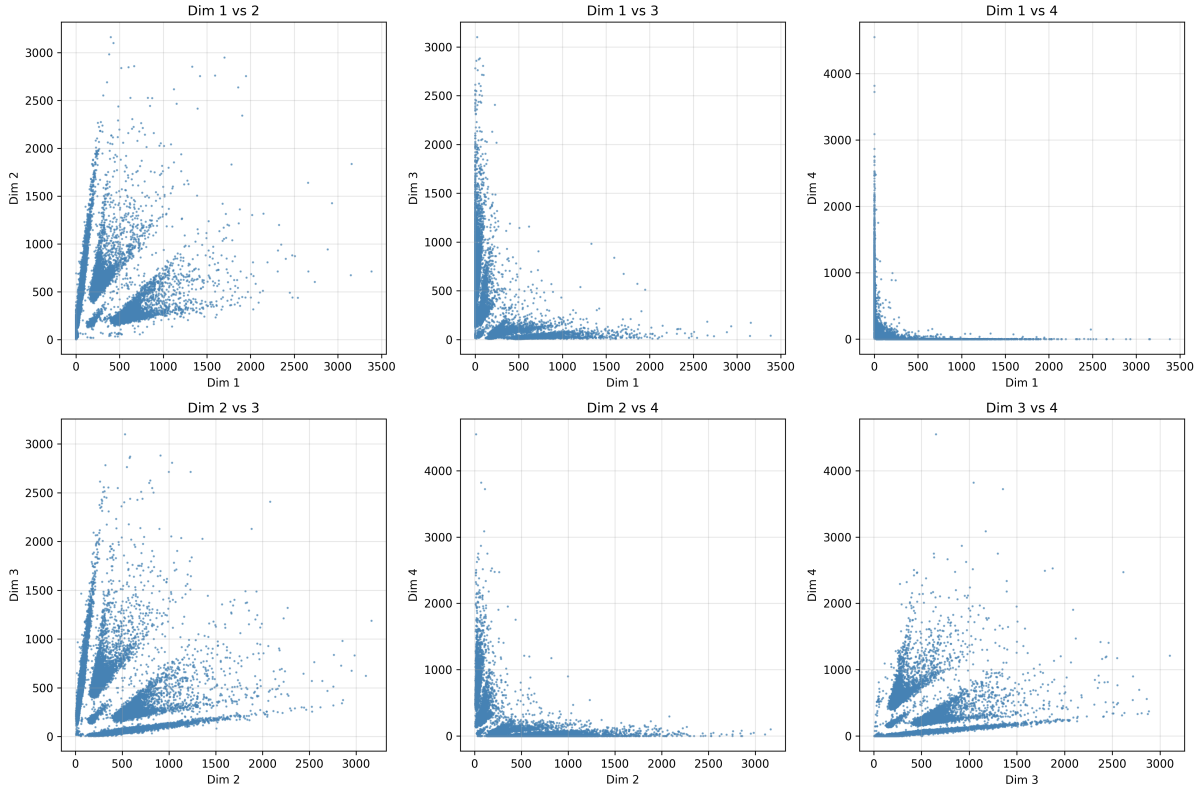
Dataset Overview

There are two types of datasets provided:

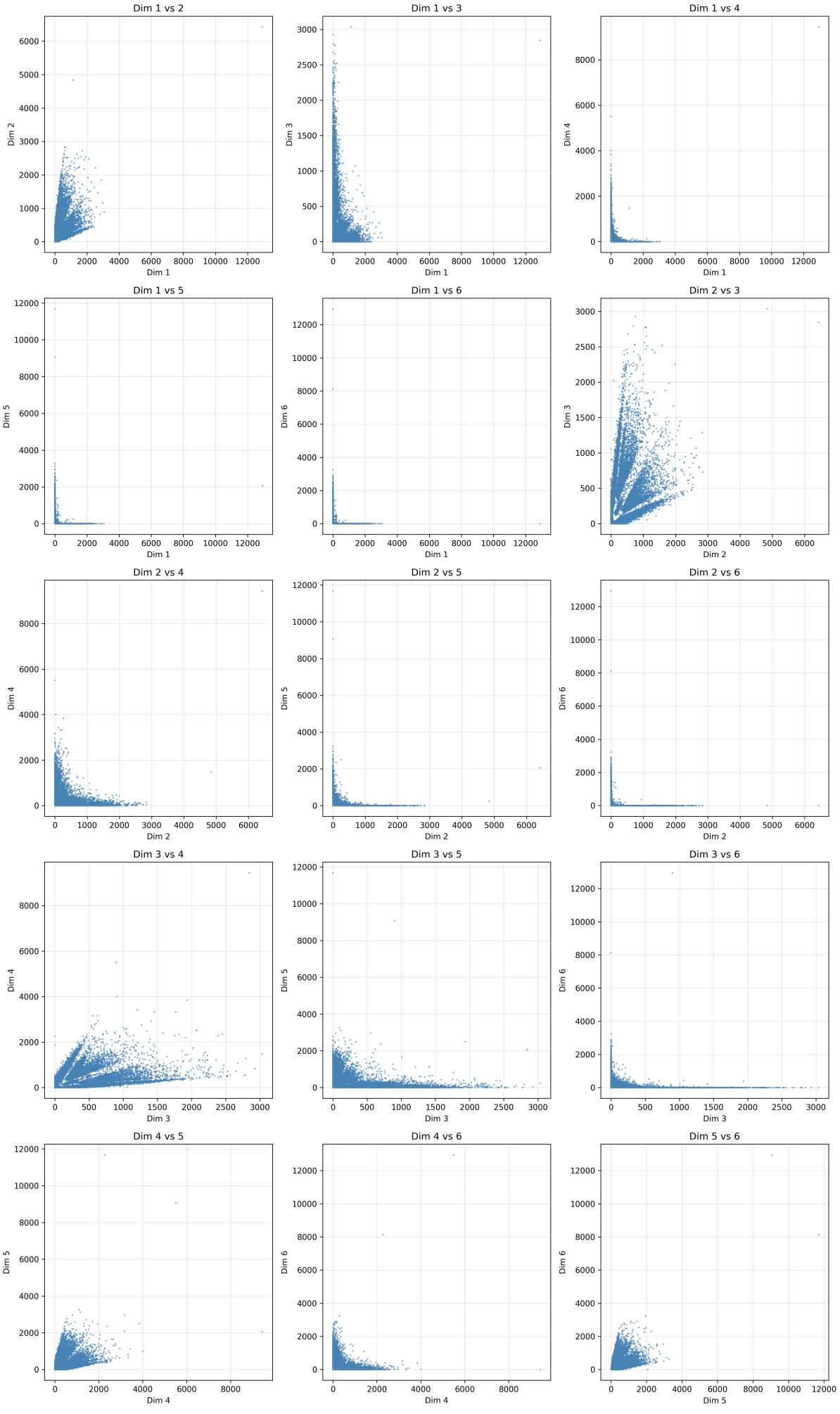
- **Public Dataset**: 4-dimensional, 49,771 entries.
- **Private Dataset**: 6-dimensional, 200,000 entries.

Each column represents the intensity recorded by a specific detector. While exact semantics of each feature are not revealed, we know that detectors are identical and values lie in a fixed range. The problem is intentionally designed to be **information-scarce**, testing the model's ability to infer structure from limited metadata.

Below is the scatter plot of both the public and private datasets:



(i) Scatter plot of public_data.csv



(ii) Scatter plot of private_data.csv

From the scatter plots, we observe that **adjacent dimensions exhibit clearer clustering structures**, suggesting that leveraging *adjacent feature interactions* might enhance clustering performance.

Experimental Pipeline (Batch Run)

We implemented a flexible grid search framework to evaluate the performance of various hyperparameter combinations.

```
# Traverse all the hyperparameter combinations
methods = ['kmeans', 'hierarchical', 'gmm', 'specialized_gmm',
'specialized_gmm_v0','adjacent_focus', 'hybrid']
scalers = ['standard', 'robust', 'quantile', 'power']
covariances = ['full', 'tied', 'diag', 'spherical']
adjacent_focuses = ['None', 'dim12', 'dim23', ...]

 # All optional flags ENABLED during grid search
flags = ['feature', 'outlier', 'refine']
```

The clustering logic is encapsulated in the `perform_clustering()` function, which follows the pipeline below:

```
def perform_clustering():
1. Load the dataset
2. Physics-aware feature engineering     # enabled via 'feature' flag
3. Apply scaling method suitable for exponential-like distributions
4. Outlier detection and handling        # enabled via 'outlier' flag
5. Clustering!! fit and predict with specified model
6. Physics-aware clusters refinement     # enabled via 'refine' flag
7. Create submission
```

Experimental Results and Final Settings

The table below summarizes the top configurations from our experiments (for the complete list, please refer to `grid_result.txt`):

Rank	ID	FMI Score	Method	Scaler	Covariance	Adjacent
1	133	0.9377	hybrid	robust	tied	None
2	120	0.9357	hybrid	standard	tied	dim34
3	117	0.9243	hybrid	standard	tied	None
3	119	0.9243	hybrid	standard	tied	dim23

Rank	ID	FMI Score	Method	Scaler	Covariance	Adjacent
8	50	0.9141	specialized_gmm_v0	standard	tied	None
9	30	0.9080	gmm	power	tied	None
11	5	0.9072	hierarchical	standard	N/A	dim12
17	2	0.8922	kmeans	robust	N/A	None
28	41	0.8236	specialized_gmm	quantile	full	None
29	69	0.8187	adjacent_focus	standard	tied	dim23

Finally, we decided to select **ID 30** (`gmm + power scaler + tied covariance`) as our final configuration for the following reasons:

- **Generalizability**: Unlike dimension-specific settings such as `dim34`, this setting avoids overfitting to the public dataset's distribution.
- **Efficiency and Reproducibility**: This setup runs faster and is easier to reproduce, demonstrating stable and consistent performance across different platforms.
- **Simplified Preprocessing**: Previous top-scoring methods with higher FMI score involved physics-aware feature engineering (`def engineer_physics_features()`) and hybrid feature creation (`def create_hybrid_features()`), which hugely increased feature dimensionality. However, this higher dimensionality likely triggered the *Curse of Dimensionality*, making it more difficult for clustering algorithms to clearly separate clusters. As a result, some cluster centers remained empty, leading to fewer than the required 4n-1 clusters.

Specifically, in our implementation, the `feature` flag (which triggers physics-aware feature engineering) is only applied for other certain methods, so it actually does **NOT** affect the `gmm` method. Furthermore, the `refine` flag performs physics-based cluster refinement based on total energy similarity, which can cause some clusters to merge and reduce the number of clusters below the expected 4n–1.

Therefore, our hyperparameter settings used for final submission is as follows:

```
methods = ['gmm']
scalers = ['power']
covariances = ['tied']
adjacent_focuses = ['None']
flags = []
```

For local testing, you can simply use the following commands:

```
python main.py public_data.csv public_submission.csv \
    --method gmm --scaler power --covariance tied
```

```
python main.py private_data.csv private_submission.csv \
    --method gmm --scaler power --covariance tied
```

> Noted that the FMI score using the exact hyperparameter settings may vary across different platforms. Local runs might yield slightly lower scores compared to those achieved on Kaggle.

Methodology and Implementation (Single Run)

This section describes the detailed steps of our final clustering pipeline, the rationale behind key design choices, and the insights gained from our various experimental attempts.

Despite its simplicity, our final pipeline achieves a high FMI score (0.9080) **while satisfying the 4n–1 clustering constraint on both public and private datasets**. The pipeline is as follows:

```
def perform_clustering():
1. Load the dataset
2. Apply scaling method with PowerTransformer
3. Clustering!! fit and predict with gmm model
4. Create submission
```

Here is some detailed explanations:

- 2. `scaler = PowerTransformer(method='yeo-johnson', standardize=True)`
  - Feature scaling is crucial since the clustering algorithms such as `gmm` and `kmeans` are sensitive to feature scale. Without proper scaling:
    - Features with larger numeric ranges dominate the distance metric or likelihood estimation.
    - This leads to biased clustering and poor performance.
  - In our experiments, we tested various scaling strategies including *StandardScaler*, *RobustScaler*, *QuantileTransformer* and *PowerTransformer*. Among these, the last one with `yeo-johnson` method performs the best.
  - *PowerTransformer* stabilize variance and make data more Gaussian-like, and this is especially beneficial for `gmm`, which assumes a Gaussian distribution for each cluster.
    - `Yeo-Johnson` method is a generalization of Box-Cox transformation and it makes highly skewed distributions more symmetric, reducing the impact of long-tailed dimensions.
  - Thus, `scaler` helps improve the convergence of `gmm`, allow the model to form more accurate and balanced clusters, reduce overfitting to high-magnitude features

- 3. `model = create_gmm_model(k_clusters, covariance_type)`
  - `gmm` models the data as a mixture of multiple Gaussian distributions, each representing a cluster. Each data point is assigned a probability of belonging to each cluster, allowing for soft clustering instead of hard assignment.
  - We carefully tuned its hyperparameters based on empirical evaluation:
    - `covariance_type='tied'` makes all clusters share the same covariance matrix, thereby reduces the number of parameters and improves model stability on limited data.
    - `init_params='kmeans'` initializes the Gaussian means using `kmeans` clustering, thus, provides a good starting point for means, speeding up EM convergence.
    - `n_init=20` runs the EM algorithm with 20 different random seeds and selects the best result based on log-likelihood.
    - `max_iter=500` allows sufficient iterations for the algorithm to converge.
    - `tol=1e-5` is the stricter convergence threshold for better precision.
    - `random_state=42` ensures reproducibility.
  - Overall, our final hyperparameter setting is as follows:

```
def create_gmm_model(k_clusters, covariance_type):
    return GaussianMixture(n_components=k_clusters,
        covariance_type=covariance_type,
        init_params='kmeans',
        n_init=20,
        max_iter=500,
        tol=1e-5,
        random_state=42
    )
```

What we attempt so far

KMeans

We initially used KMeans, which already achieved a reasonable FMI score of around 0.88 despite its simplicity. However, we ultimately sought a model with better generalizability.

GMM

We transitioned to GMM due to its probabilistic nature and reputation for superior flexibility. However, initial attempts with default hyperparameters performed poorly. After extensive tuning, we observed significant variance in performance across configurations, and finally achieved a FMI of 0.9080, while also consistently producing exactly 4n–1 clusters, satisfying the core requirement.

More Complete Pipeline

We also experimented with several pipeline components, including Principal Component Analysis (PCA), the `outlier` flag, and the `refine` flag. However, none of these additions resulted in significant improvements in clustering performance.

Adjacent Focus

Following the TA's suggestion, we analyzed adjacent dimension scatter plots, and observed clearer cluster boundaries in adjacent dimension pairs. This motivated us to test adjacent-focused clustering by looping through all dim_ij combinations. However, the results did not meet our expectations. For example, under 'hybrid + standard + tied' setting, the ranking was dim34 > none > dim23, showing no consistent winner. Considering the potential drop in generalizability, we ultimately excluded adjacent focus from the final GMM setup.

Feature Augmentation

Our final attempt was to explore physics-aware feature engineering and hybrid feature creation which significantly boosted FMI scores across many configurations (as evidenced by our top-ranked runs). These techniques added domain-specific interpretability, but drastically increased feature dimensionality. This likely triggered the Curse of Dimensionality, making clustering unstable and always producing fewer than 4n–1 clusters due to empty components. Even after attempting dimensionality reduction, the tradeoff between score gains and structural compliance was not favorable.

Final Decision - GMM as above

Despite its apparent simplicity, this model achieved good enough performance, stable generalizability, and structural correctness (consistently producing 4n–1 clusters). This outcome demonstrates that with carefully chosen preprocessing steps and well-tuned hyperparameters, a clean and minimal pipeline can outperform more complex methods involving heavy feature engineering, especially in clustering tasks constrained by strict structural requirements.

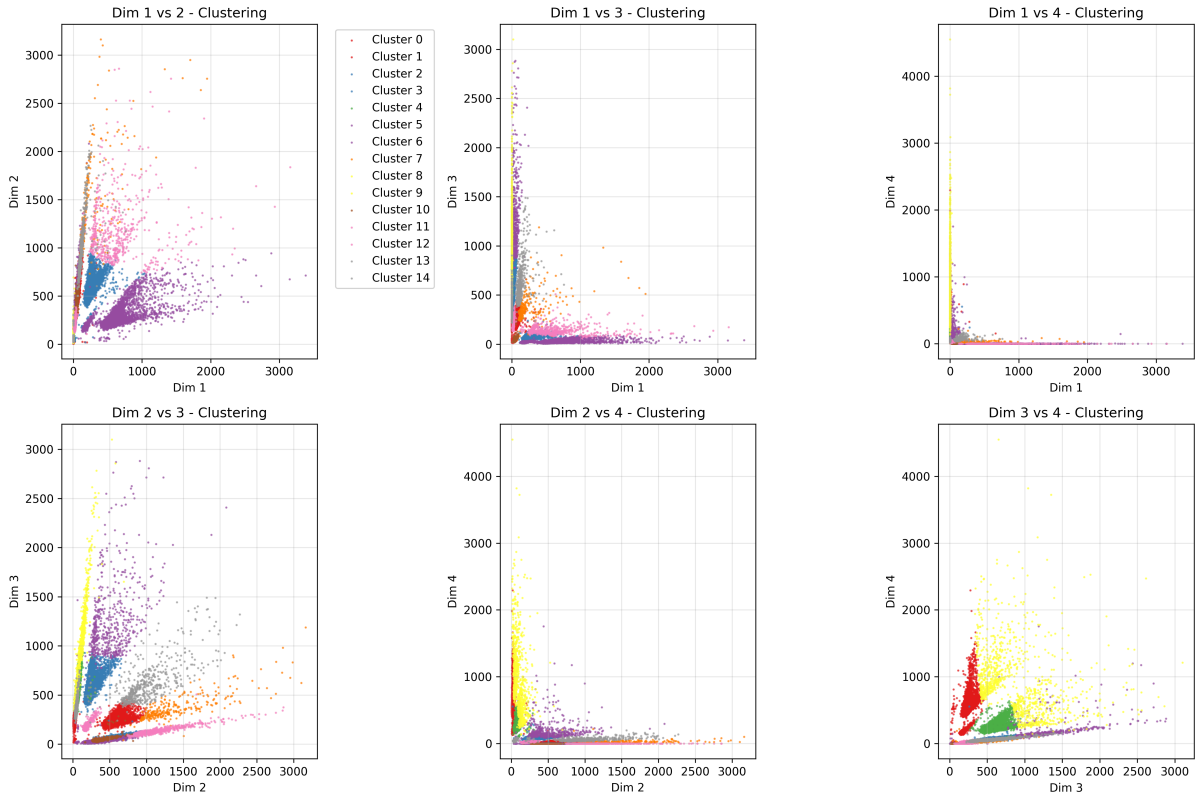
Results and Discussion

With the final hyperparameter setting mentioned above, we run this on **Kaggle with NVIDIA T4 GPU ×2**, achieving an **FMI score of 0.9080** on the public dataset. Importantly, this configuration successfully produced exactly **4n−1 clusters** on both the public and private datasets, fully satisfying the structural requirement.

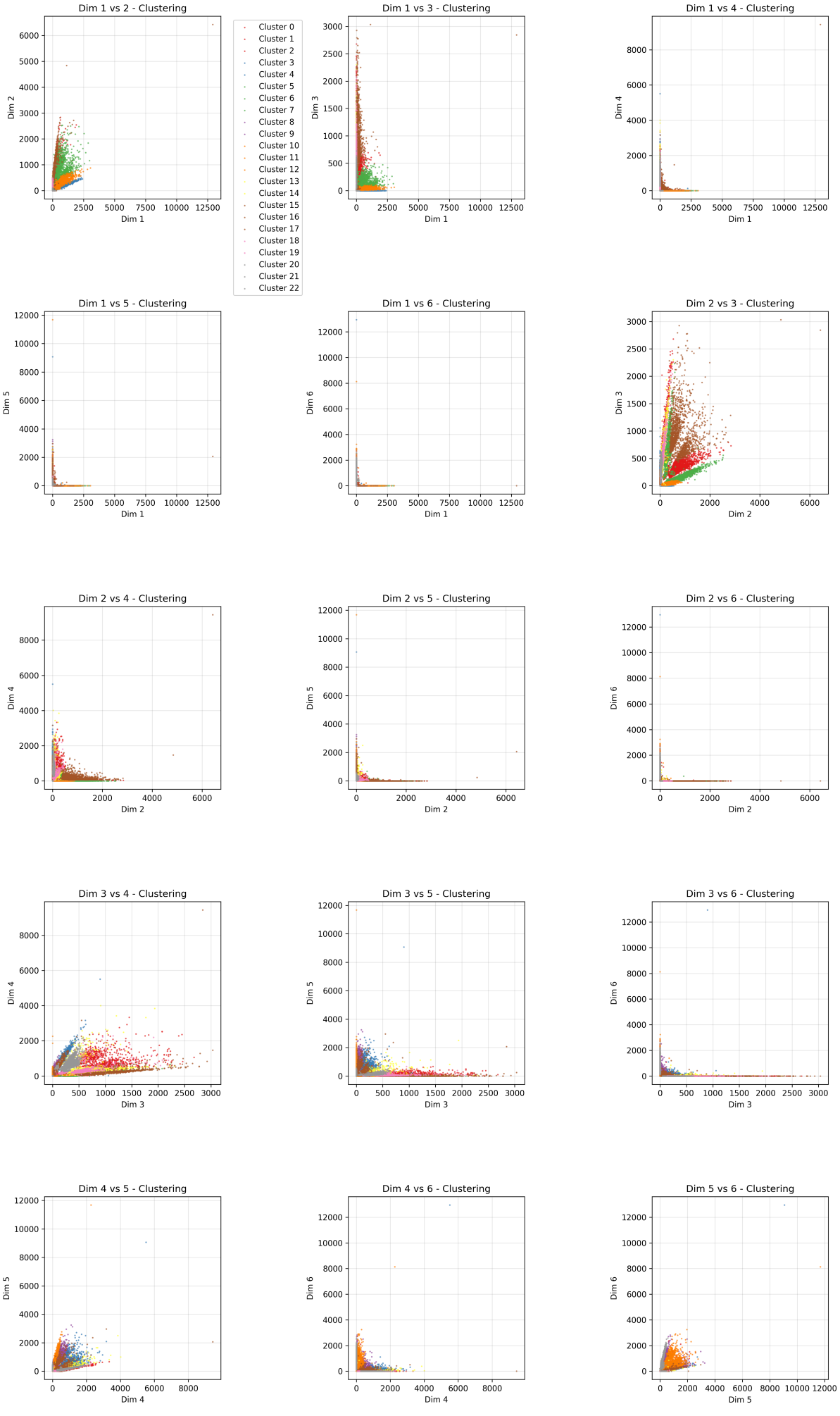
> For full reproducibility, please refer to our GitHub Link. Before executing, please carefully read the `README.md`, which provides a clear breakdown of each folder, script, and execution command.

According to the scatter plots below, the clustering results on the public dataset exhibit clear and well-separated clusters, while the results on the private dataset are relatively less structured. This highlights the significant impact that **feature dimensionality** and **dataset size** can have on the performance of clustering models.

Below are the clustering visualizations for both datasets:



(i) Clustering results on the public dataset



(ii) Clustering results on the private dataset

Conclusion

This project turned out to be significantly more challenging and rewarding than initially expected. While our early baseline using `KMeans` already achieved an FMI score around **0.88**, a great deal of additional effort went into experimenting with alternative methods, feature engineering, hyperparameter tuning, and structural constraints enforcement.

Through systematic exploration and deliberate tradeoffs, we only improved the final score to **0.9080**, while also ensuring the clustering output satisfied the **4n–1** constraint. Along the way, we gained practical experience in applying probabilistic models like **GMM**, understanding the effects of **dimensionality**, and evaluating model generalization on unseen data.

In summary, the final solution may appear simple, but it is the result of extensive experimentation, informed reasoning, and iterative refinement.