Machine Learning in Computational Biology
Master of Science – "Data Science and Information Technologies"
Academic year 2020-2021
Prof. Elias Manolakos

Usupervised Learning

Dimensionality Reduction, Clustering & Visualization

Aspasia Vozi

Contents

1 Introduction

We have given five single-cell synthetic datasets, all generated from the same model, that represent synthetic single cell RNA sequencing data. To that end, each dataset comprise 200 cells and each cell is characterized by 200 gene expression features. Our goal is to analyze the dataset in a pipeline that accepts as input a dataset and performs dimensionality reduction of the gene expression data and clustering of the dimensionality reduced data. Finally, the results should be visualized in an intuitive manner in order to help us approximate the actual distribution that generated the given datasets.

For the purpose of this assignment we used the `Python` programming language, and we have implemented the code in the provided `Jupyter notebook`. All the dependencies required to run the notebook, are provided in Appendix A (Section 4).

2 Methods

2.1 Datasets Overview

First, we should perform some initial analysis and calculate some basic statistics for the five given datasets. To get started, we loaded a random dataset, using the Python `pandas` and `numpy` libraries, in order to visualize the data and perform some basic preprocessing and analysis.

Detection of outliers

For each dataset we plot a heatmap representing the expression of each gene in each cell, as well as a bar plot of the top 10 expressed genes in the dataset. By looking at the results we can see that although there are some genes per dataset that are highly expressed, none of them seems to be highly expressed across all five datasets. Moreover, the heatmaps appear to be somewhat noisy and there is neither an apparent conclusion from these figures just by looking at them, nor we can easily filter out some genes by consider them outliers or irrelevant features.

Before we start performing some kind of dimensionality reduction we should first standardize our datasets in order to have zero mean and unit variance. Thus, we create a simple function using the `StandardScaler` provided by the Python `sklearn` package that does the job for us.

All the heatmaps are provided in the Appendix B (Section 4).

2.2 Dimensionality Reduction

Dimensionality reduction, as suggested by its name, aims to reduce the number of dimensions in the data. Regarding the single cell RNA sequence analysis field, dimensionality reduction is essential and very useful since different genes may be correlated, if they are participating in the same biological process. Thus, we can compress the feature space into a lower dimensional space. Computationally this is a very important step in further

analysis, since it reduces the calculations needed and the noise of the data. Furthermore, it enables the visualization of the analysis for those of use who are not mathematicians and can only perceive up to 3 dimensions.

2.2.1 Principal Component Analysis (PCA)

Principal components analysis (PCA) consists one of the most well known and broadly used dimensionality reduction methods. It discovers axes in high-dimensional space that capture the largest amount of variation in the data. Initially, an axis is chosen so that, we can move all of our observations (cells) onto this axis by the shortest possible path. The variance captured by this axis is defined as the variance across the observations along that line. In PCA, the first axis (or "principal component", PC) is chosen such that it captures the greatest variance across observations. The next PC is chosen such that it is orthogonal to the first and captures the greatest remaining amount of variation, and so on.

By definition, the top PCs are the ones that capture the dominant factors of heterogeneity in the data set. Thus, we can perform dimensionality reduction using PCA, and perform further analysis using only the top PCs. This approach is based on the properties of the theoretical background of PCA, that a low-rank approximation formed from the top PCs is the optimal approximation of the original data for a given matrix rank.

Before performing PCA, as well as the other dimensionality reduction methods, it is important to standardize our data, as mentioned in the previous section. Since PCA seeks to maximize the variance represented in each successive principal component, it is important to normalize or standardize features before performing the PCA transformation. Otherwise, some of the variance detected by PCA could simply be due to the difference in magnitude of individual variables.

The main objective when we perform PCA is to decide the number of the components (PCs) that we are going to keep in order to perform the rest of the analysis. To that end, we used the following two methods as presented below:

Choosing the number of components

1. **Cumulative Variance Criterion:** One way to decide regarding the number of components that should be used for further analysis, is to retain all PCs until the percentage of total variation explained reaches some threshold. For example, we might retain the top set of PCs that explains 80% or 95% of the total variation in the data.

   In Figure 1, the cumulative variance, of all the components of one of the datasets is presented. We have included the respective diagrams for all the provided datasets in Appendix A. As depicted in the Figure 1, ≈ 110 components are required in order to achieve the 95% of the cumulative variance. Moreover, it is shown that almost half, ≈ 65, of the components are required to achieve 80% of the cumulative variance.

   Using this method one can simply select the number of components per dataset. Although, keeping a high number of PCs may better explain the variance, it also adds to the model complexity and maintains noise in our reduced dataset.

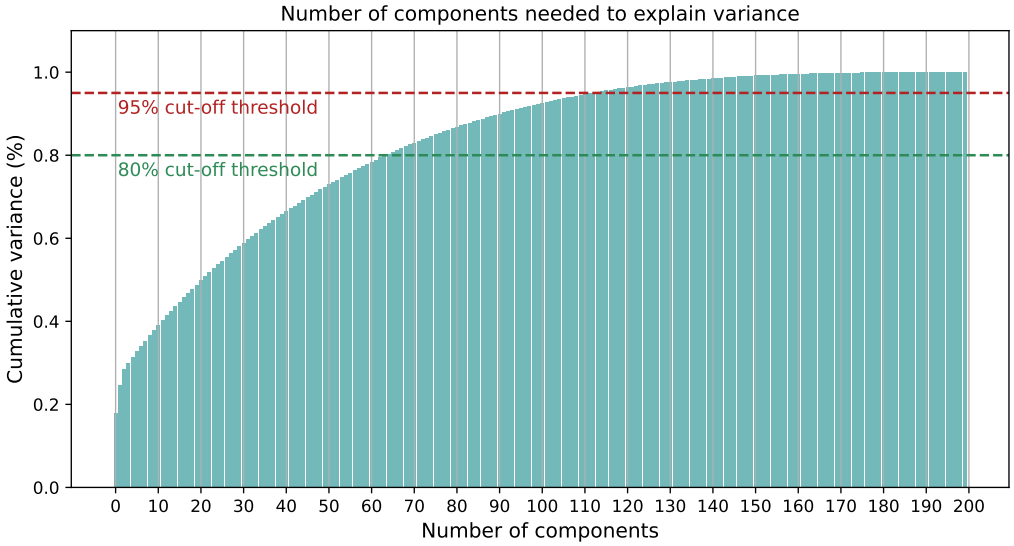

Figure 1: Cumulative Variance explained for Dataset 2.

2. **Individual Variance Criterion:** Another option for choosing the appropriate number of PCs involves identifying the elbow point in the percentage of variance explained by successive PCs (individual variance of a PC). This refers to the "elbow" in the curve of a scree plot. A scree plot is a line plot of the eigenvalues of the PCs form the PCA. This assumption is based on the fact that each of the top PCs should explain much more variance than the remaining PCs.

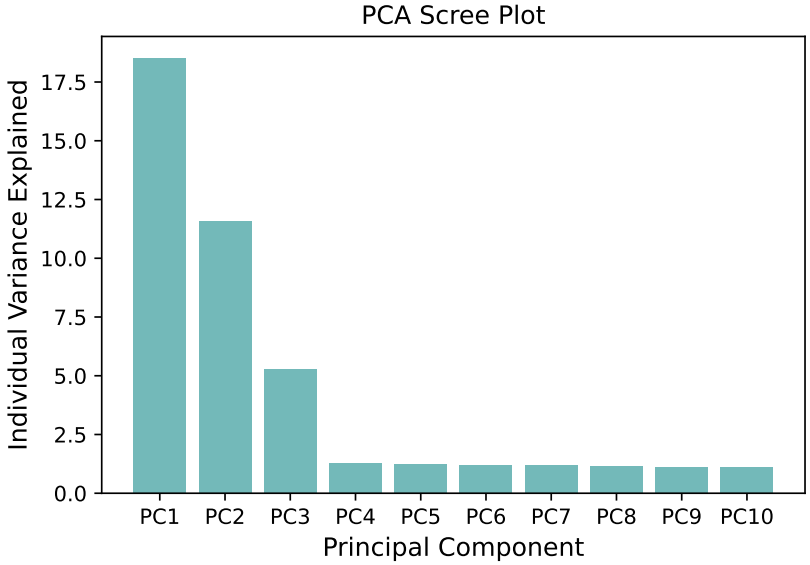

Figure 2: Individual Variance Expained

From a practical perspective, the use of the elbow point tends to retain fewer PCs compared to the previous method. As it is depicted in Figure 2, after the fourth or fifth PC, each PC does not add a satisfying percentage of the data variance.

From these results, we can assume that an appropriate number of components that we can use to perform the rest of our analysis is between 2–5. However, a more extensive evaluation regarding the number of components is carried out in the following sections.

We have created a simple function for running PCA into a given dataset, using the `PCA` function provided by the Python `sklearn` package.

2.2.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

T-Distributed Neighbor Embedding (t-SNE)[2], is a non linear dimensionality reduction method that was introduced in 2008 and visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. In brief t-SNE works as follows:

- Given a set of N multidimensional observations $X=(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N})$, t-SNE first computes probabilities $p_{i|j}$, that are proportional to the similarity of objects $\mathbf{x_i}$ and $\mathbf{x_j}$, as follows. For $i \neq j$ define:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

  where $p_{i|i} = 0$ and we know that $\sum_j p_{j|i} = 1$ for all i.

- As explained in the paper[2] : "The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i."

- The bandwidth of the Gaussian kernels σ_i is set in such a way that the perplexity of the conditional distribution equals a predefined perplexity. As a result, the bandwidth is adapted to the density of the data, that is, smaller values of σ_i are used in denser parts of the data space.

- t-SNE aims to learn a d-dimensional map $\mathbf{y}_1, \ldots, \mathbf{y}_N$ (with $\mathbf{y}_i \in \mathbb{R}^d$ $\mathbf{y}_i \in \mathbb{R}^d$) that reflects the similarities p_{ij} as well as possible. To that end, it measures similarities q_{ij} between two points in the map $\mathbf{y}_i$ and $\mathbf{y}_j$, using a similar approach. Specifically, for $i \neq j$, define q_{ij} as

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

  where $q_{ii} = 0$.

- The locations of the points $\mathbf{y}_i$ in the map are determined by minimizing the Kullback–Leibler divergence of the distribution P from the distribution Q, that is:

$$\mathrm{KL}\left(P \parallel Q\right) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

  The minimization of the Kullback–Leibler divergence with respect to the points $\mathbf{y}_i$ is performed using gradient descent. The result of this optimization is a map that reflects the similarities between the high-dimensional inputs.

The performance of t-SNE is fairly robust under different settings of the perplexity. The most appropriate value depends on the density of the data. It is said, that a larger / denser dataset requires a larger perplexity. Typical values for the perplexity are suggested to be in range between 5 and 50. Perplexity is a measure for information that is defined as 2 to the power of the Shannon entropy. In t-SNE, the perplexity may be viewed as a knob that sets the number of effective nearest neighbors.

In order to visualize the effect of the perplexity parameter, we create a simple function for running t-SNE into a given dataset, using the `TSNE` function provided by the Python `sklearn` package. We applied the t-SNE dimensionality reduction to the sample dataset and plot the results for 2 and 3 components in order to get an intuition about the results. It should be mentioned that t-SNE, can run for up to 3 dimensions, as explained in `sklearn` documentation. This is in agreement with the bibliography that suggests that t-SNE works better as a visualization tool.

In Figure 3, we note that in the 3D space, t-SNE tends to map the data in very compact structures with uniformly distributed points for all the datasets. We observe that for a `perplexity` of 15, we get the better visualization for our data. Thus, we decided to proceed using the following parameters for t-SNE:

- **number of components: 2**
- **preplexity: 15**

2.2.3 Uniform Manifold Approximation and Projection (UMAP)

The uniform manifold approximation and projection (UMAP) method [3] is an alternative to t-SNE for non-linear dimensionality reduction. UMAP works very similar to t-SNE, use try to arrange data in low-dimensional space, trying to preserve relationships between neighbors in high-dimensional space. However, the two methods are based on a different theory.

In order to construct the initial high-dimensional graph, UMAP builds a fuzzy simplicial complex. This is a representation of a weighted graph, with edge weights representing the likelihood that two points are connected. To determine connectedness, UMAP extends a radius outwards from each point, connecting points when those radius overlap. Choosing the radius is critical, since too small value may lead to small, isolated clusters, while too large values will connect everything together. UMAP overcomes this challenge by choosing a radius locally, based on the distance to each point's nth nearest neighbor. UMAP then makes the graph "fuzzy" by decreasing the likelihood of connection as the radius grows. Finally, by stipulating that each point must be connected to at least its closest neighbor, UMAP ensures that local structure is preserved in balance with global structure. Once the high-dimensional graph is constructed, UMAP optimizes the layout of a low-dimensional analogue to be as similar as possible, the same way as t-SNE does.

In general, while both algorithms tend to exhibit strong local clustering and group similar categories together, UMAP is able to clearly separate the sub-clusters of a cluster from each other.

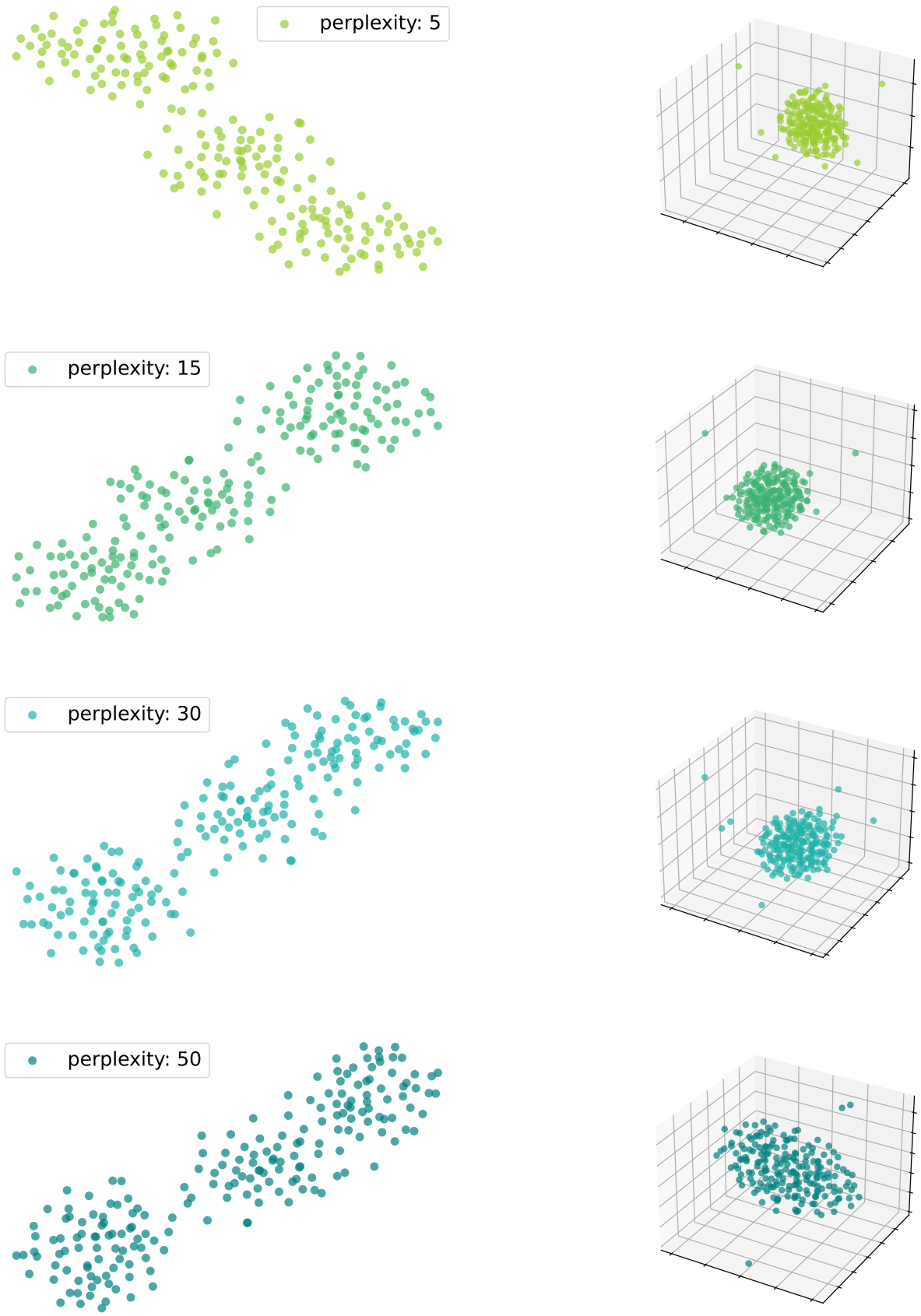

Figure 3: t-SNE for different perplexity values using 2 components (left) and 3 components (right), on dataset 2.

Similar to t-SNE, in order to visualize the effect of the parameters, we created a simple function for running UMAP into any given dataset, using the `umap` Python package.

Parameter Selection
In general, the two most commonly used parameters are `n_neighbors` and `min_dist`, which are effectively used to control the balance between local and global structure in the final projection.

1. **Number of neighbors:** This parameter refers to the number of approximate nearest neighbors used to construct the initial high-dimensional graph. It controls how UMAP balances local versus global structure. We know that low values will push UMAP to focus more on local structure by constraining the number of neighboring points considered when analyzing the data in high dimensions, while high values will push UMAP towards representing the big-picture structure while losing fine detail.

   In Figure 4, we present the effect of different values of the `n_neighbors` parameter in the predicted structures, for 2 and 3 components. We note that as the number of neighbors increases, the predicted structures, as expected, tend to unify and become more compact. We can conclude that for **number of neighbors=15**, we get the best results for both 2 and 3 components.

2. **Minimum Distance:** This parameter controls how tightly UMAP colocates points together. In general, low values lead to more tightly packed embeddings, while larger values make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure.

In Figure 5, we present the effect of different values of `min_dist` in the predicted structures for 2 and 3 components. We note that as the value of the minimum distance between the neighbors increases, the predicted structures, as expected, tend to unify and become more compact. We can conclude that for **minimum distance between 0 and 0.1**, we get the best results for both 2 and 3 components. Note that, in 3 dimensions, the best structure seem to be obtained for `min_dist = 0.5`.

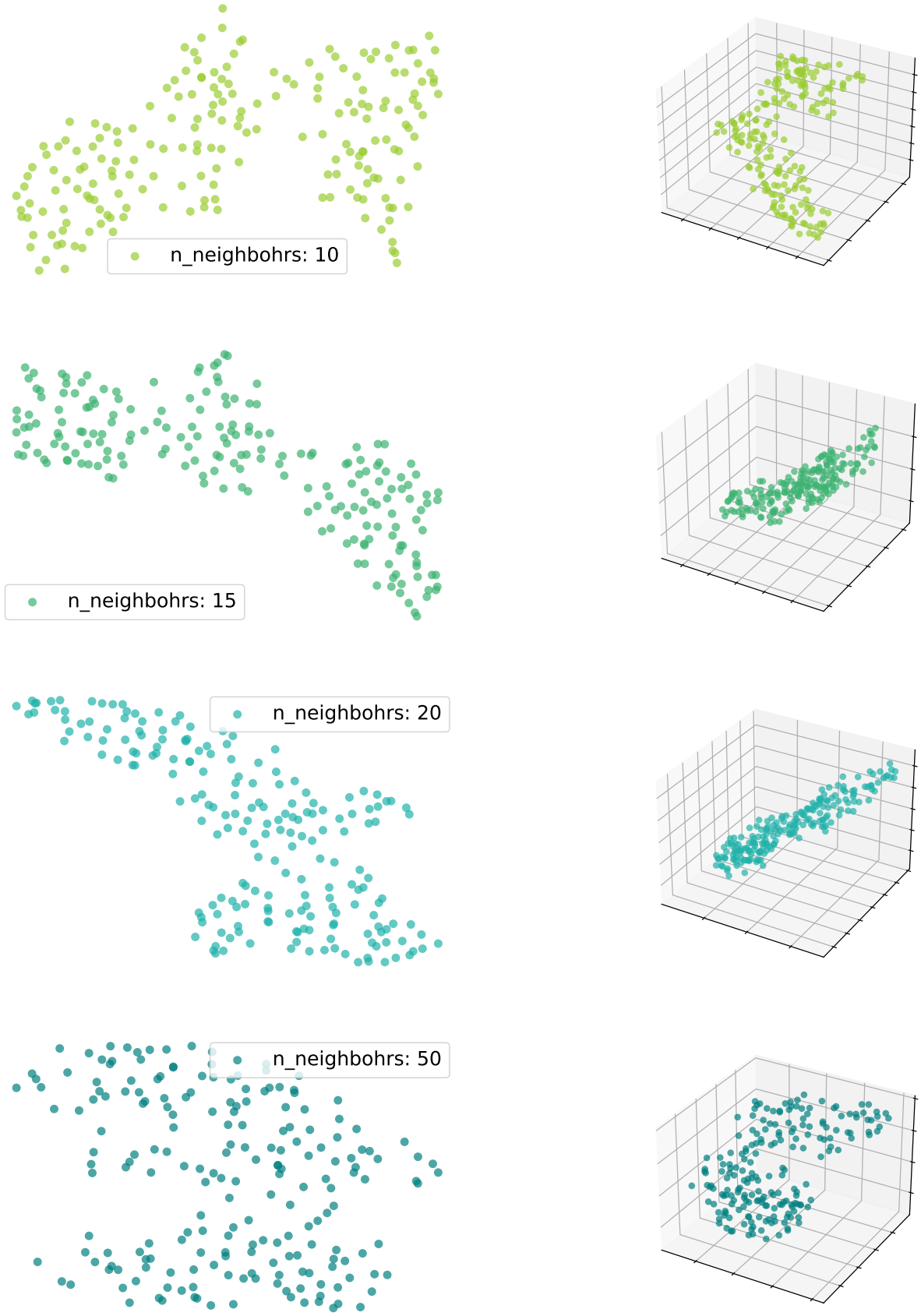

Figure 4: UMAP for different number of neighbors, `min_dist=0.1` using 2 components (left) and 3 components (right), on dataset 2.

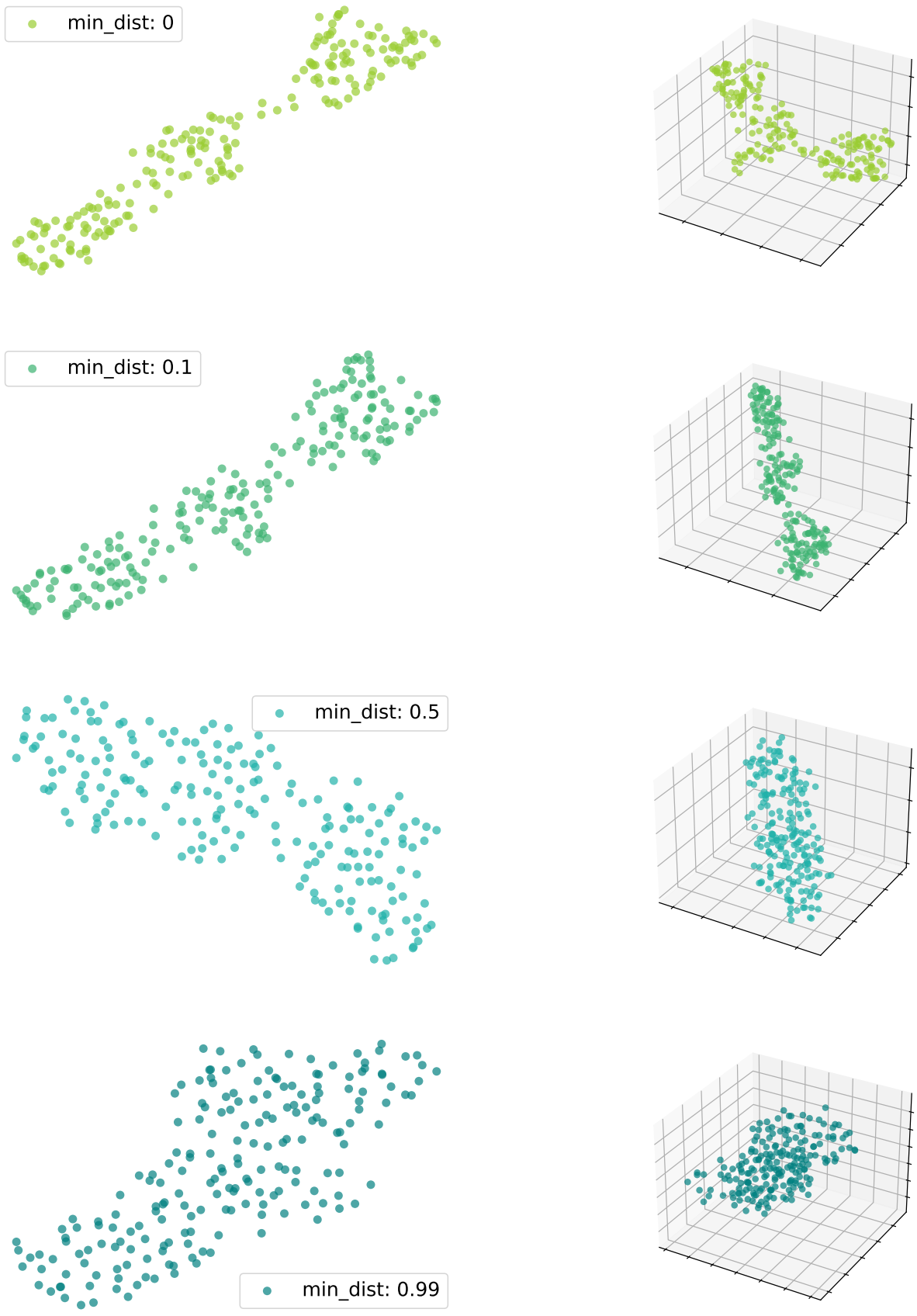

Figure 5: UMAP for different minimum distance values, `n_neighbors=15` using 2 components (left) and 3 components (right), on dataset 2.

2.3 Clustering using Gaussian Mixture Modeling

Superpositions formed by taking linear combinations of Gaussian distributions, can be formulated as probabilistic models known as mixture distributions. We therefore define a superposition of K Gaussian densities of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

which is called a mixture of Gaussians. Each Gaussian density is called a component of the mixture and the parameters π_k are called the mixing coefficients. By using a sufficient number of Gaussians, and by optimizing their means and covariances, as well as the coefficients π_k in the linear combination, almost any continuous density can be approximated to arbitrary accuracy.

Thus, a Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions having unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data, as well as the centers of the latent Gaussians.

Given a dataset $\mathbf{X} = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$, one way of finding the values of the parameters is to use maximum likelihood. Due to the presence of the summation over k inside the logarithm, the derivation is complex, and as a result, the maximum likelihood solution for the parameters does not have a closed-form analytical solution. One approach to maximizing the likelihood function is to employ the Expectation-Maximization technique.

We create a simple function for running GMMs for a given dataset, using the `gmm` function of the Python package `sklearn` ((Section 4)). The custom `gmm` function takes as input the reduced dataset, the number of states we want to predict and four types of covariance matrix.

2.3.1 BIC Criterion

We use the BIC criterion [1] in order to evaluate our models, as well as further investigate the impact that will have in our models of we choose different number of components.

Mathematically BIC can be defined as

$$BIC = \ln(n)k - 2\ln(\hat{L})$$

where n is the number of data points, k is the number of free parameters to be estimated and $\hat{L}$ is he maximized value of the likelihood function of the model under examination.

The models were tested using corresponding BIC values. In general, lower BIC value indicates lower penalty terms hence a better model.We plot the BIC values of each model, for different number of states. Moreover we calculate the delta BIC score as follows:

$$\Delta BIC = BIC_m - BIC^*$$

where BIC_m is the BIC of the model under examination and BIC^* is the optimum BIC. In this way, we try not to be greedy in terms of choosing the model with the best BIC score. By inspecting the ΔBIC score instead of only looking at the BIC score, we can decide the optimal number of states with respect to the more stable model, by taking into consideration also the models that don't have the optimal BIC score but a good score.

2.3.2 Silhouette Score

An alternative for evaluating the quality of the Gaussian mixture model is to employ the *Silhouette* score [4], which is calculated using the mean intra-cluster distance $\alpha(i)$ and the mean nearest-cluster distance $\beta(i)$ for each sample i as follows:

$$s(i) = \frac{\beta(i) - \alpha(i)}{\max\{\alpha(i), \beta(i)\}}$$

where $\beta(i)$ refers to the distance between the sample i and the nearest cluster that the sample is not a part of. After computing the silhouette score for each sample, the mean of these values is calculated as an overall indicator of the quality of the clustering.

Intuitively, values near 0 indicate overlapping clusters, while negative values generally indicate that samples has been assigned to the wrong cluster, as a different clusters are more similar.

3 Results

In order to decide the best model for the given datasets, we designed a pipeline that should be followed in order to understand the data and approximate the actual number of clusters (true distribution) that generated the observed datasets. In this section we present the results obtained from an exploratory data analysis using three dimensionality reduction techniques and the Gaussian mixture models over various parameter configurations.

3.1 PCA & GMM

As discussed in Section 2.2.1, we need to decide the best number of components for PCA. Moreover, for fitting a Gaussian mixture model (GMM) on the data, we need to specify the type of the covariance matrix, as well as the number of clusters (states). In order to decide on these parameters we ran a swipe over a range of values for these parameters and scored each model using both the BIC criterion and the Silhouette score (see Sections 2.3.1 and 2.3.2).

The results for the BIC score are presented in Figure 6. Each row of the matrix refers to a specific dataset, while each column to a specific covariance matrix type. Every bar plot in the matrix plots the BIC score against the number of states (clusters) used by the GMM. Each bar color refers to a different number of components for the PCA. The results indicate that the lowest BIC score is achieved by simpler models in terms of PCA components, as they achieve the lowest BIC score regardless of the number of clusters and covariance matrix type.

On the other hand, Figure 7 depicts the ΔBIC instead of BIC. The results in this case suggest that the full covariance matrix type seems to achieve its best BIC score using a smaller number of components than every other covariance matrix type, since most of them require more than 6 components.

These observations are also supported by Figure 8, that plots the Silhouette score instead of the BIC score. The higher scores are achieved by using 2 components in every covariance matrix type. Moreover, the score seems to achieve its maximum value for fewer components (generally less than 6) in all cases.

Finally, in Figure 9 and Figure 10 we plot a matrix of the GMM clustering results using only the full covariance matrix. For each dataset we mark using a green color the clustering that achieved the best BIC score. The results show that using 2 components leads to more consistent results than using 3 components. For the three first datasets the best model requires 5 clusters, for the fourth dataset, requires 4 clusters, while for the last dataset requires 7 clusters. On the other hand, using 3 components we obtain 4 clusters for the first and fourth dataset, 6 clusters for the second dataset and 8 clusters for the third and last datasets. However, the differences in BIC score between these cases are not that large. One interesting observation is that in the last dataset there is a tendency to prefer a larger number of clusters, regardless of the number of components we choose. That may be justified by the odd shape of the dataset in comparison to the other four.

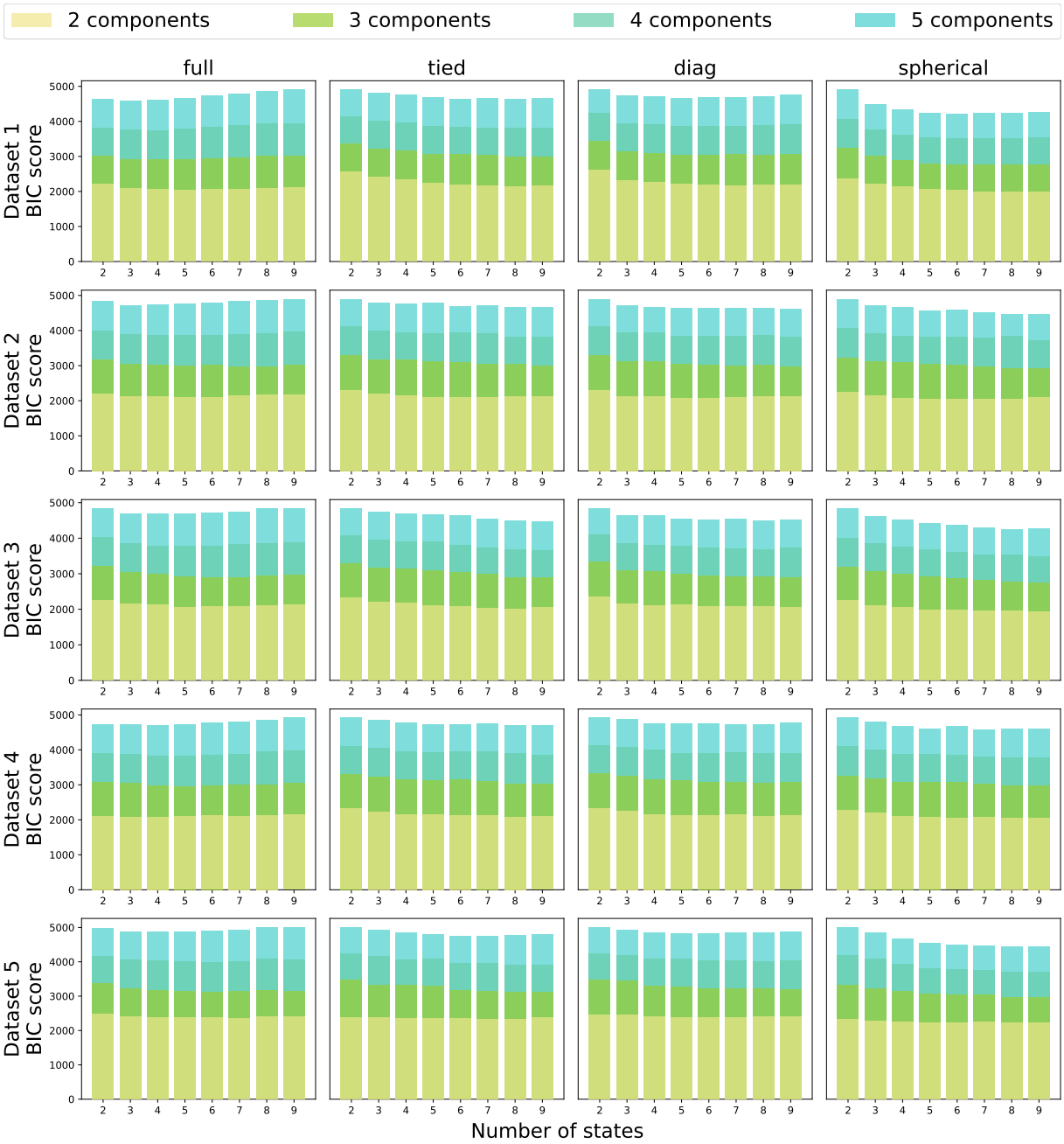

Figure 6: *BIC* scores for different number of components, for each of covariance matrix for each dataset, using **PCA for dimensionality reduction** and GMM for different numbers of states.

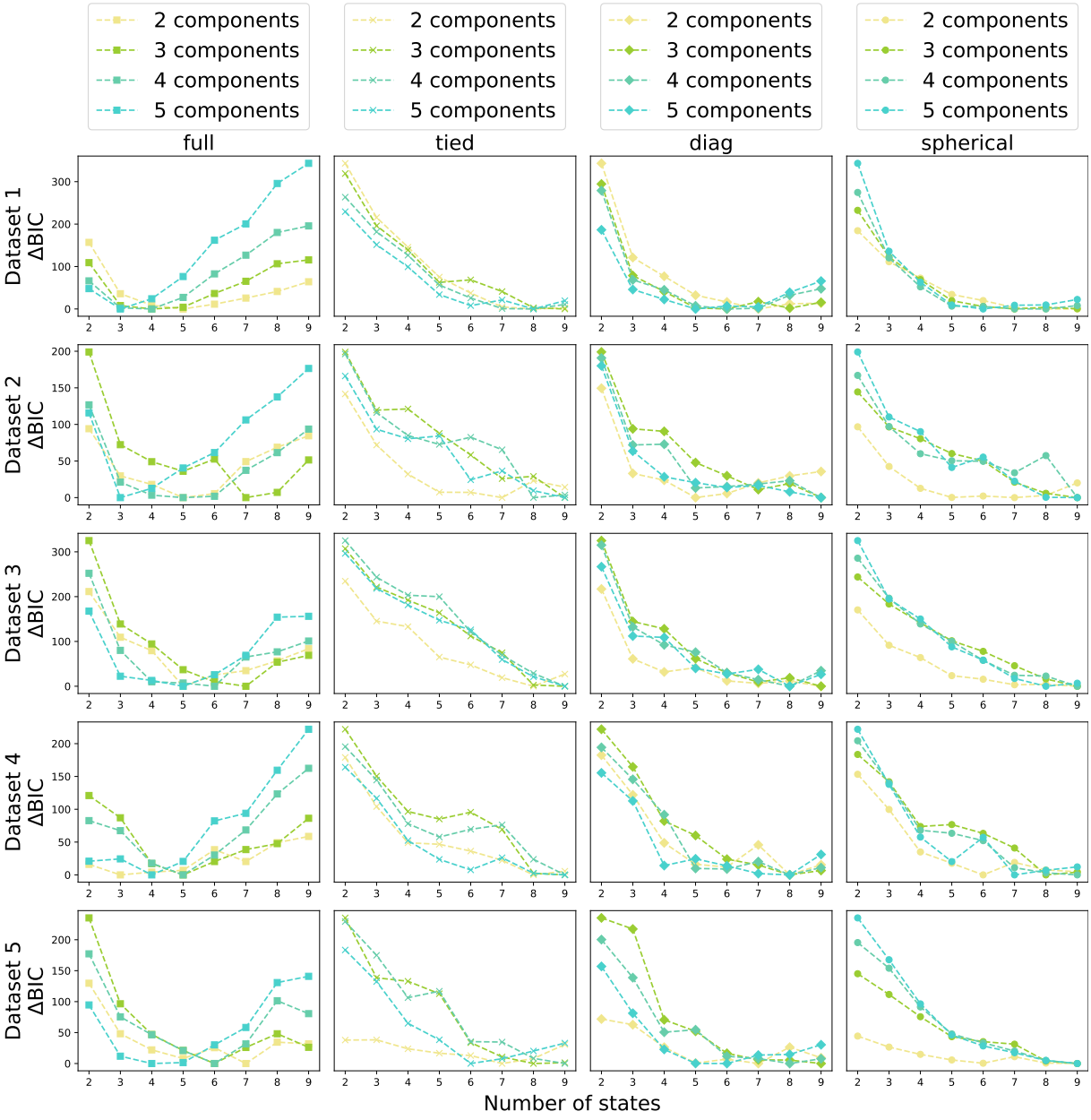

Figure 7: ΔBIC scores for different number of components, for each of covariance matrix for each dataset, using **PCA for dimensionality reduction** and GMM for different numbers of states.

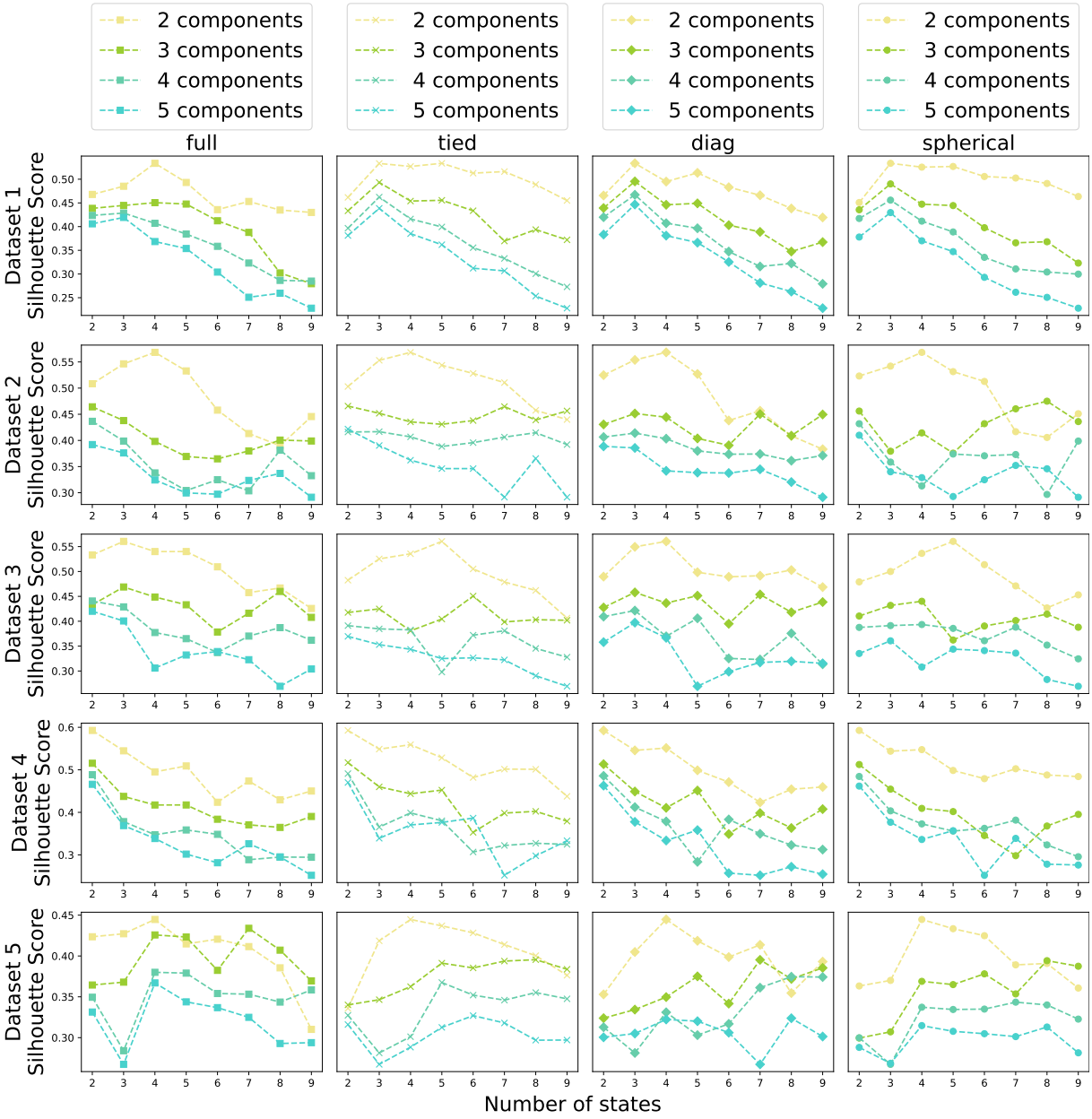

Figure 8: *Sihlouette* scores for different number of components, for each of covariance matrix for each dataset, using **PCA for dimensionality reduction** and GMM for different numbers of states.

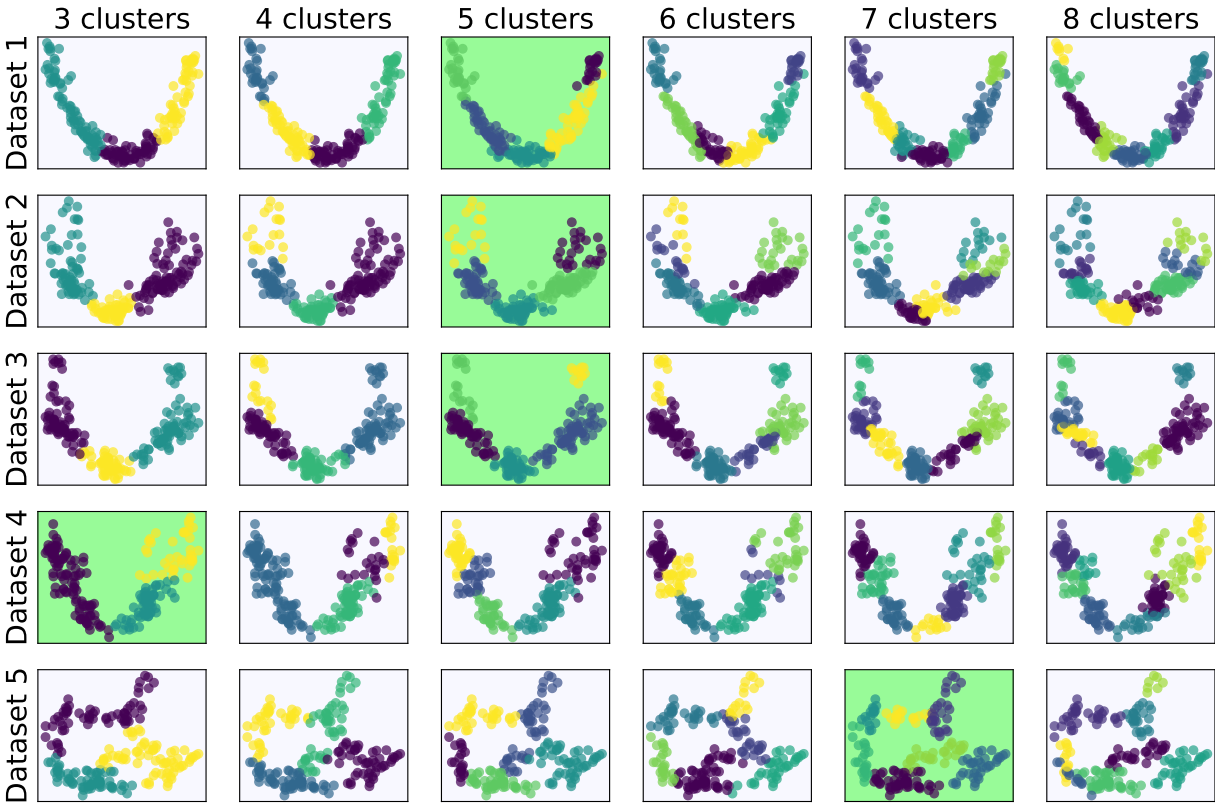

Figure 9: For each dataset we plot the clusters obtained from GMM model, using as input 2 components from **PCA**. With green color we denote the best GMM model per dataset, with respect to the best BIC score.

Figure 10: For each dataset we plot the clusters obtained from GMM model, using as input 3 components from **PCA**. With green color we denote the best GMM model per dataset, with respect to the best BIC score.

3.2 t-SNE & GMM

As discussed in Section 2.2.2, t-SNE works better for lower dimensions, thus we will use only two components to reduce the features of the initial datatsets. In 2.2.2, we have also discussed regarding the perplexity of t-SNE. To that end, we have concluded that the best value for the perplexity parameter is around 30. For fitting a Gaussian mixture model (GMM) on the data, we also need to specify the type of the covariance matrix, as well as the number of clusters (states). In order to decide on these parameters we ran a swipe over a range of values for these parameters and scored each model using both the BIC criterion and the Silhouette score (see Sections 2.3.1 and 2.3.2).

The results for the BIC score are presented in Figure 11. Each row of the matrix refers to a specific dataset, while each column to a specific covariance matrix type. Every bar plot in the matrix plots the BIC score against the number of states (clusters) used by the GMM. Each bar color refers to the number of components for the t-SNE.

On the other hand, Figure 12 depicts the ΔBIC instead of BIC. In comparison to the previous pipeline, the results in this case suggest that not only the full covariance matrix type achieves its best BIC score using a smaller number of components, but the same stands also for the almost all the other covariance matrix type. They all seem to have the best BIC scores for 3 states.

These observations are also supported by Figure 13, that plots the Silhouette score instead of the BIC score.

In Figure 14 we plot a matrix of the GMM clustering results using only the full covariance matrix. For each dataset we mark using a green color the clustering that achieved the best BIC score. For the two datasets the best model requires 2 clusters (not presented in this plot), for two dataset, requires 3 clusters, while for the last dataset requires 4 clusters. The last dataset also here requires the larger number of states, as observed in the previous section as well. In general, the t-SNE pipeline seems to have the worst performance so far, with respect to the BIC score, since it achieves twice the value of the respective PCA pipeline BIC scores. Finally, its performance can be attributed to the predicted structures.

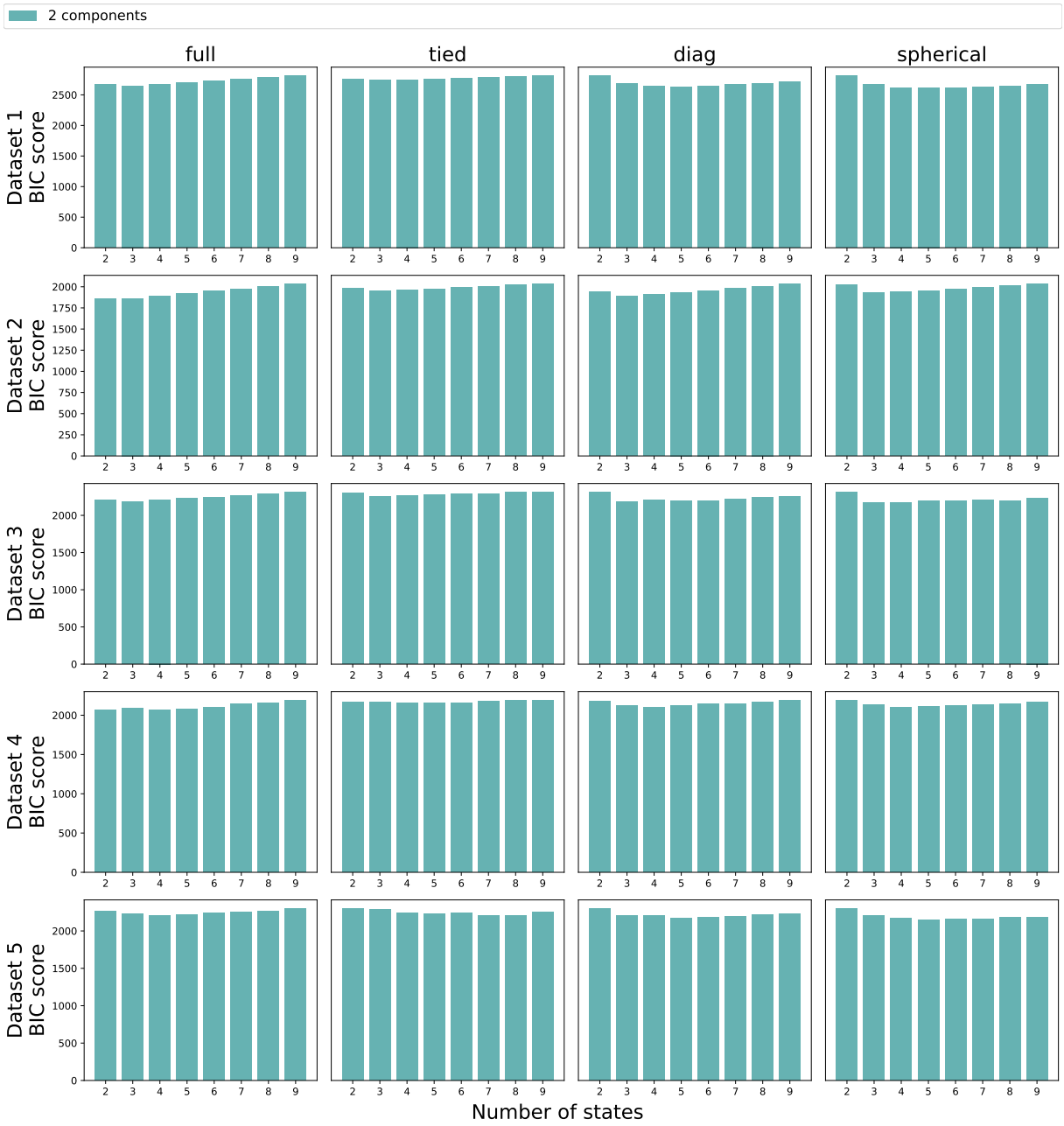

Figure 11: BIC scores for 2 components, for each of covariance matrix for each dataset, using **t-SNE for dimensionality reduction** and GMM for different numbers of states.

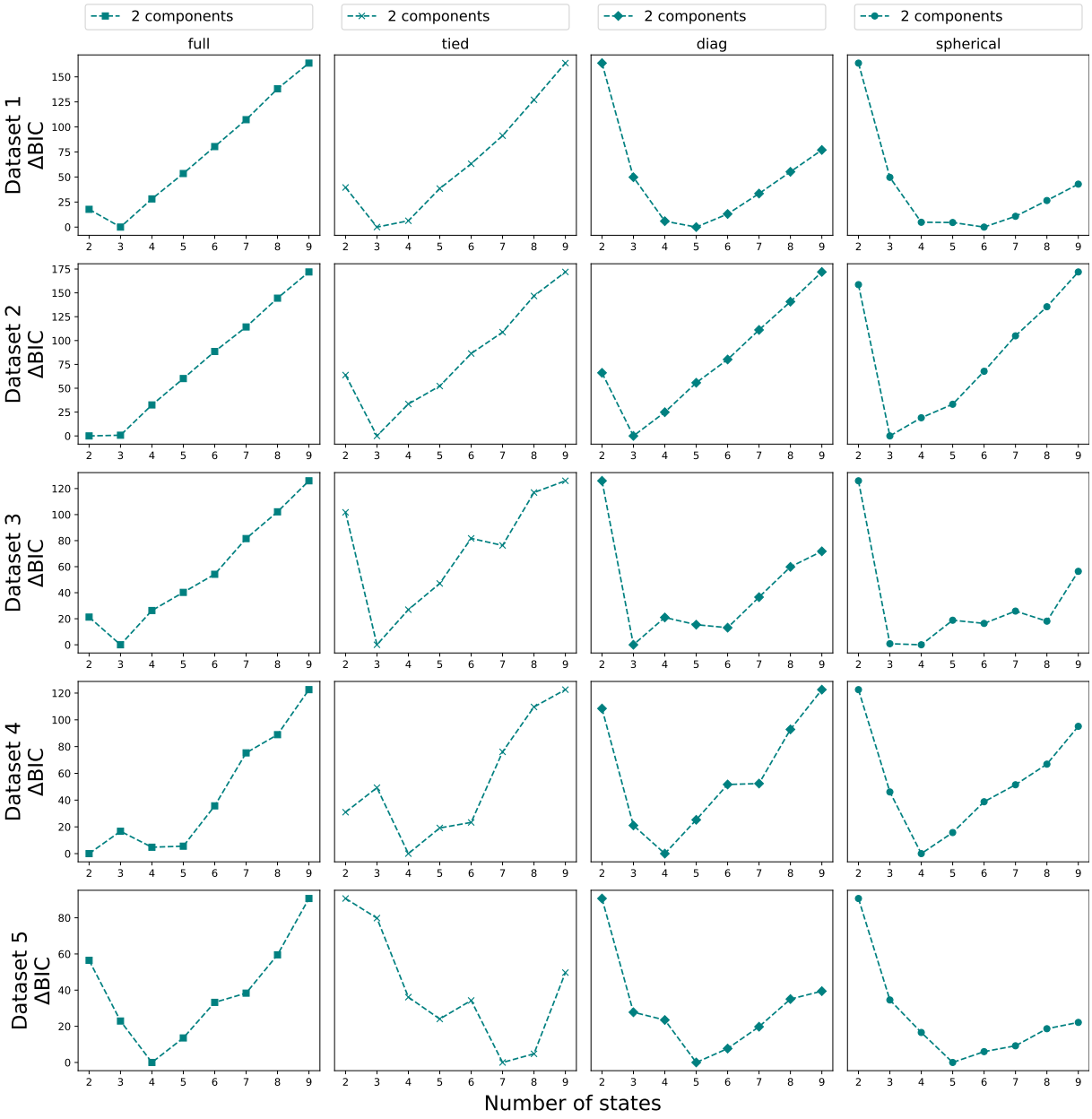

Figure 12: ΔBIC scores for different number of components, for each of covariance matrix for each dataset, using **t-SNE for dimensionality reduction** and GMM for different numbers of states.

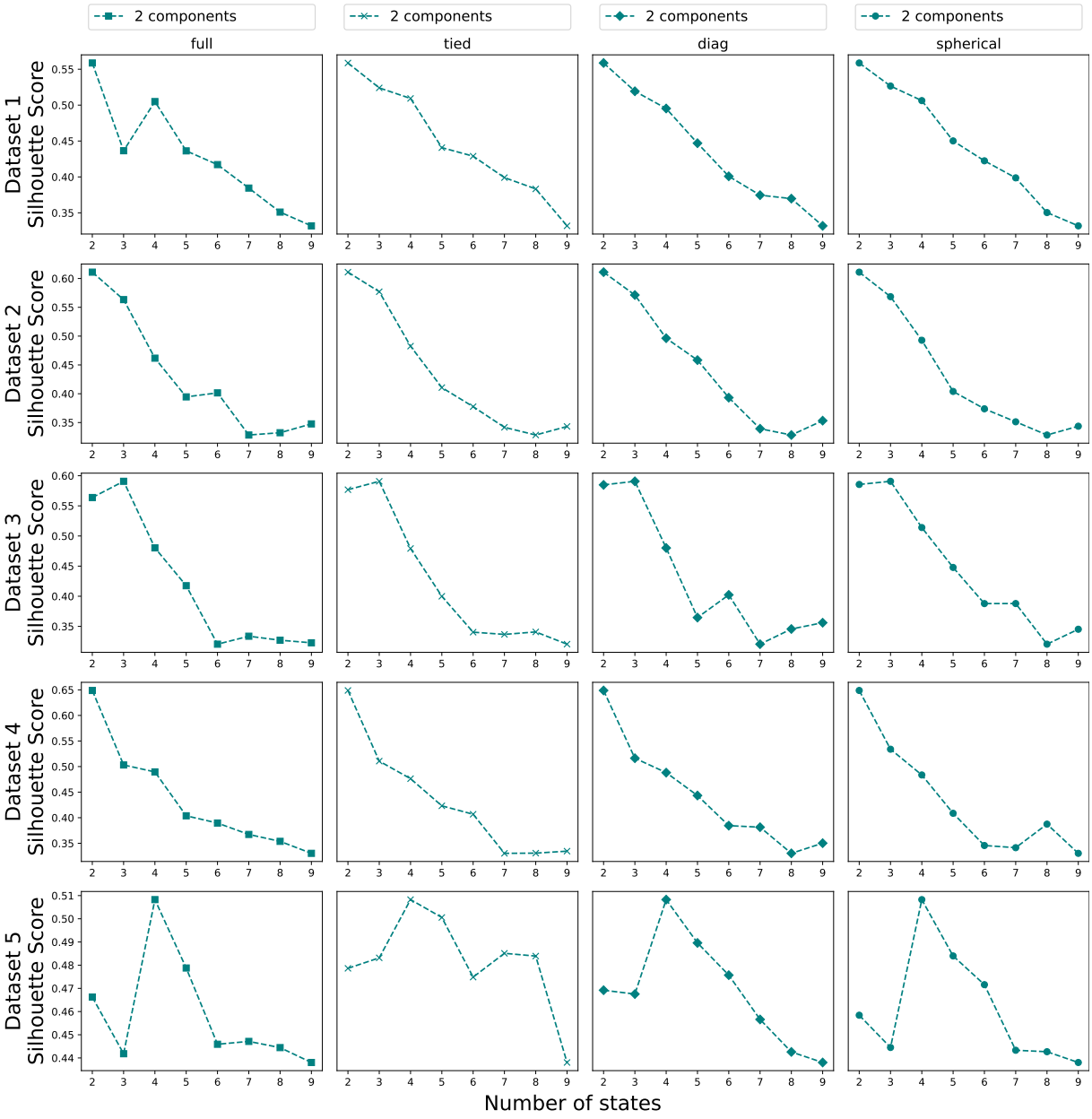

Figure 13: *Sihlouette* scores for different number of components, for each of covariance matrix for each dataset, using **t-SNE for dimensionality reduction** and GMM for different numbers of states.

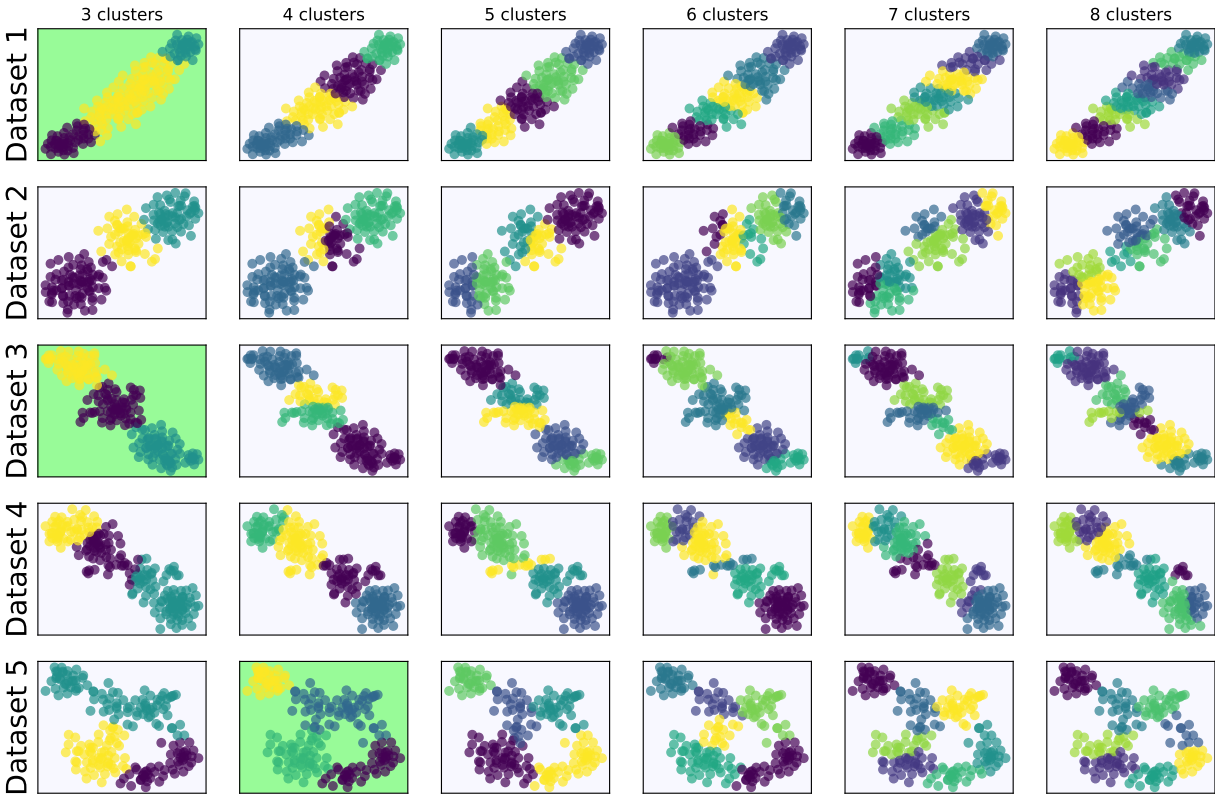

Figure 14: For each dataset we plot the clusters obtained from GMM model, using as input 2 components from **t-SNE**. With green color we denote the best GMM model per dataset, with respect to the best BIC score.

3.3 UMAP & GMM

Similar to t-SNE and PCA, we performed the same exploratory analysis for the UMAP method. However, since UMAP can perform reduction in any number of dimension, in contrast to t-SNE, and we did not have an intuitive method, similar to PCA in order to get an appropriate range of components to evaluate, we picked a range from 2 to 8. Figure 15 presents the BIC score for all different components. The results indicate a preference for 8 components in some datasets, while for others 2-3 components seem to suffice. However, the largest preference towards 8 components is present in dataset 5 that has an odd shape with respect to the rest of the datasets.

Figure 16 and Figure 17 present the results for ΔBIC and Silhouette score. These results suggest once again that the full covariance matrix seem to need less clusters in order to achieve a good clustering compared with the other covariance matrices. The Silhouette results show that although the BIC score seem to be better on some datasets when choosing a larger number of components, even using 2 or 3 components has comparable quality in terms of the Silhouette score.

Finally, Figure 18 and Figure 19 depict the clustering results for 2 and 3 components respectively. These results show that 3 to 5 clusters are required in the 2-dimensional case for datasets 1 through 4, while the fifth dataset still seem to require 7 clusters in the best model. The same holds in 3 dimensions, where 3 or 4 clusters suffice for the first four datasets, whule the fifth requires 6 clusters.

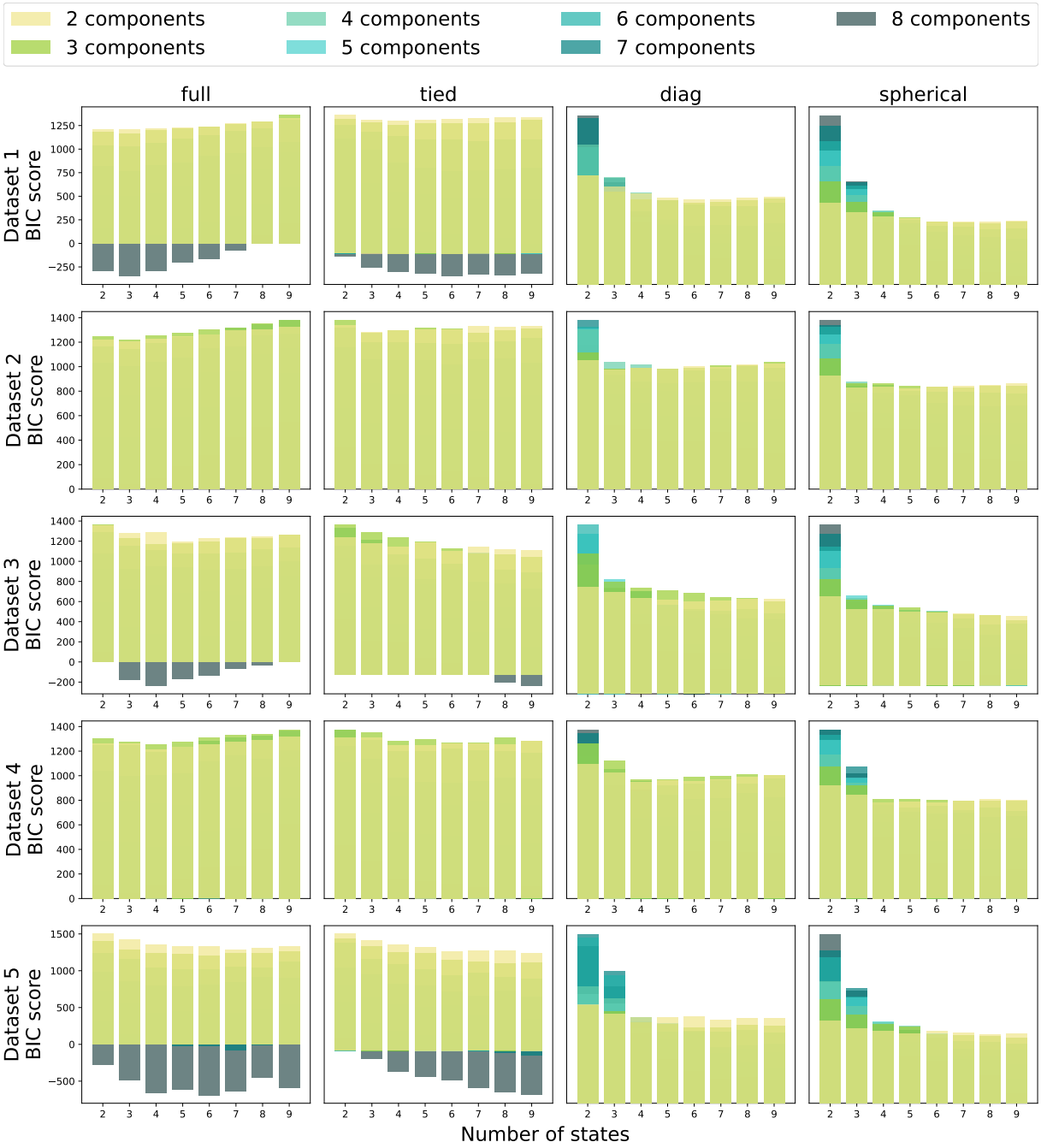

Figure 15: BIC scores for different number of components, for each of covariance matrix for each dataset, using **UMAP for dimensionality reduction** and GMM for different numbers of states.

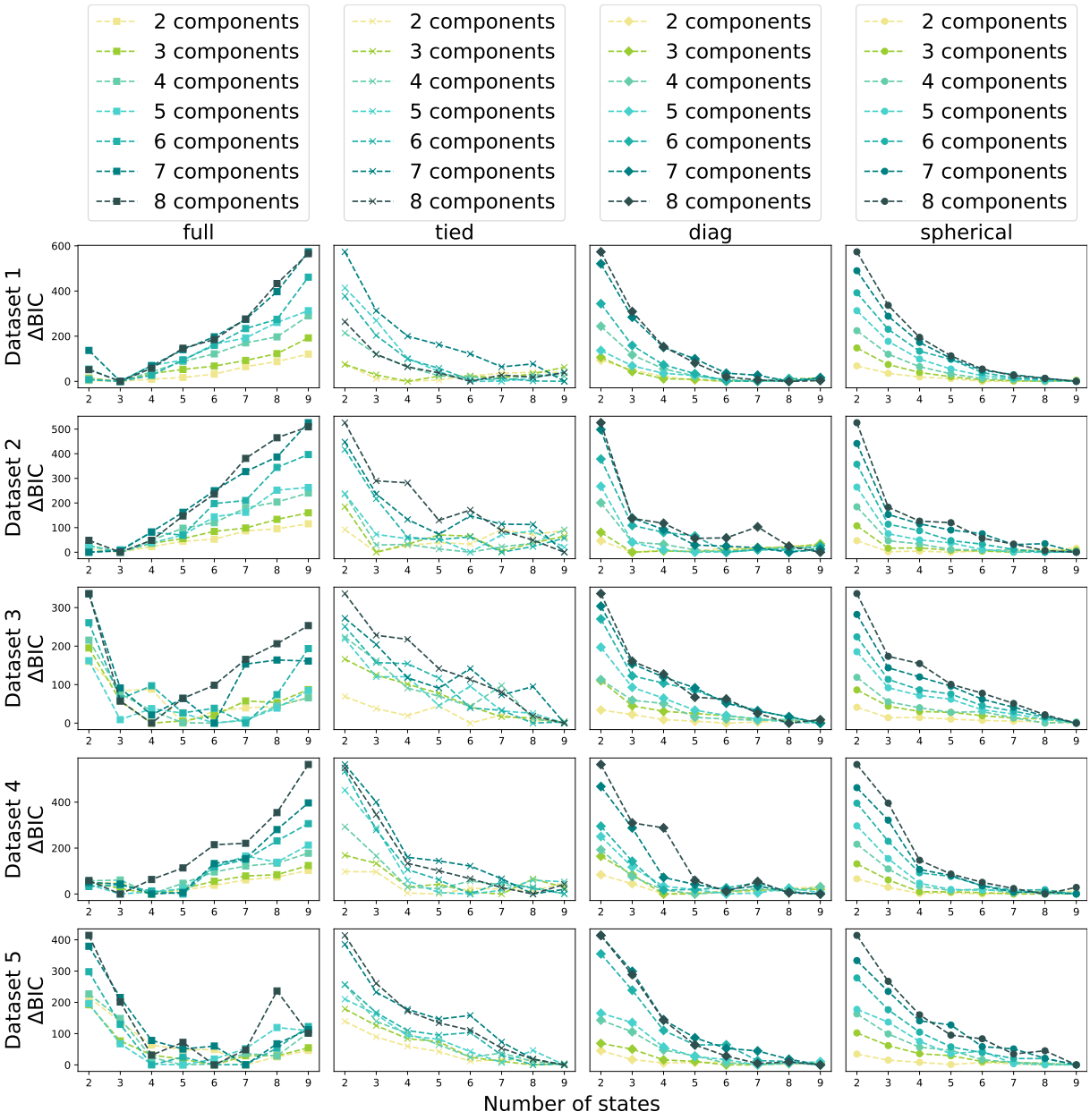

Figure 16: ΔBIC scores for different number of components, for each of covariance matrix for each dataset, using **UMAP for dimensionality reduction** and GMM for different numbers of states.

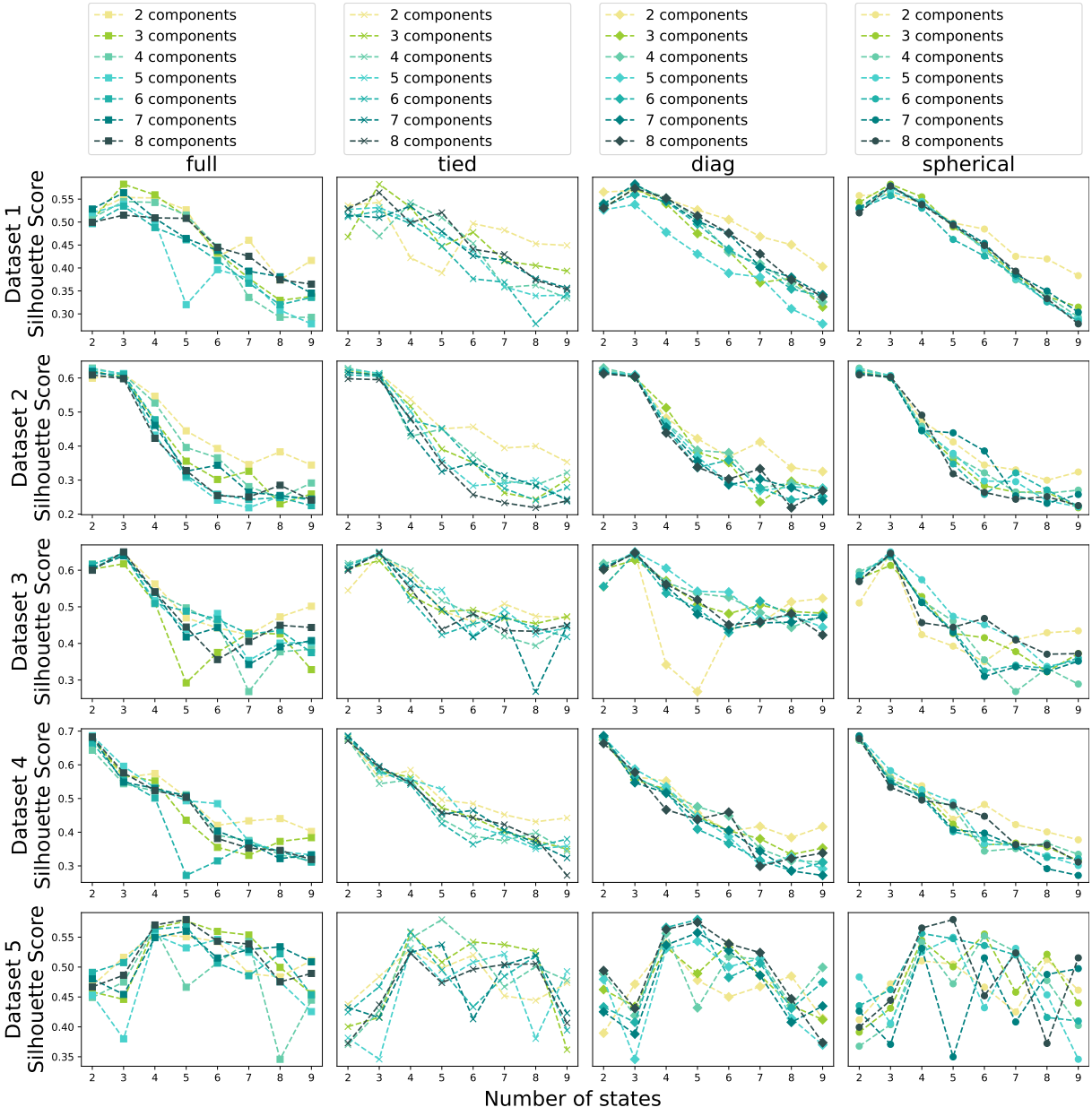

Figure 17: *Sihlouette* scores for different number of components, for each of covariance matrix for each dataset, using **UMAP for dimensionality reduction** and GMM for different numbers of states.

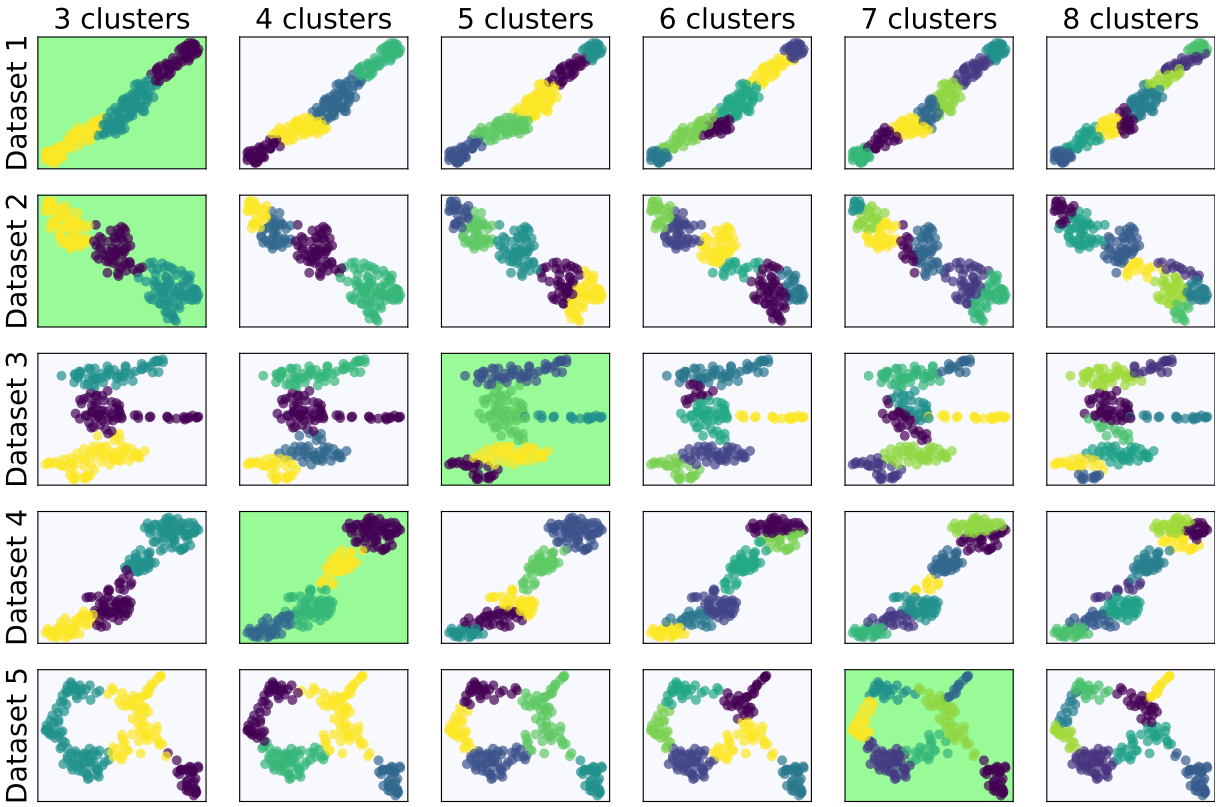

Figure 18: For each dataset we plot the clusters obtained from GMM model, using as input 2 components from **UMAP**. With green color we denote the best GMM model per dataset, with respect to the best BIC score.

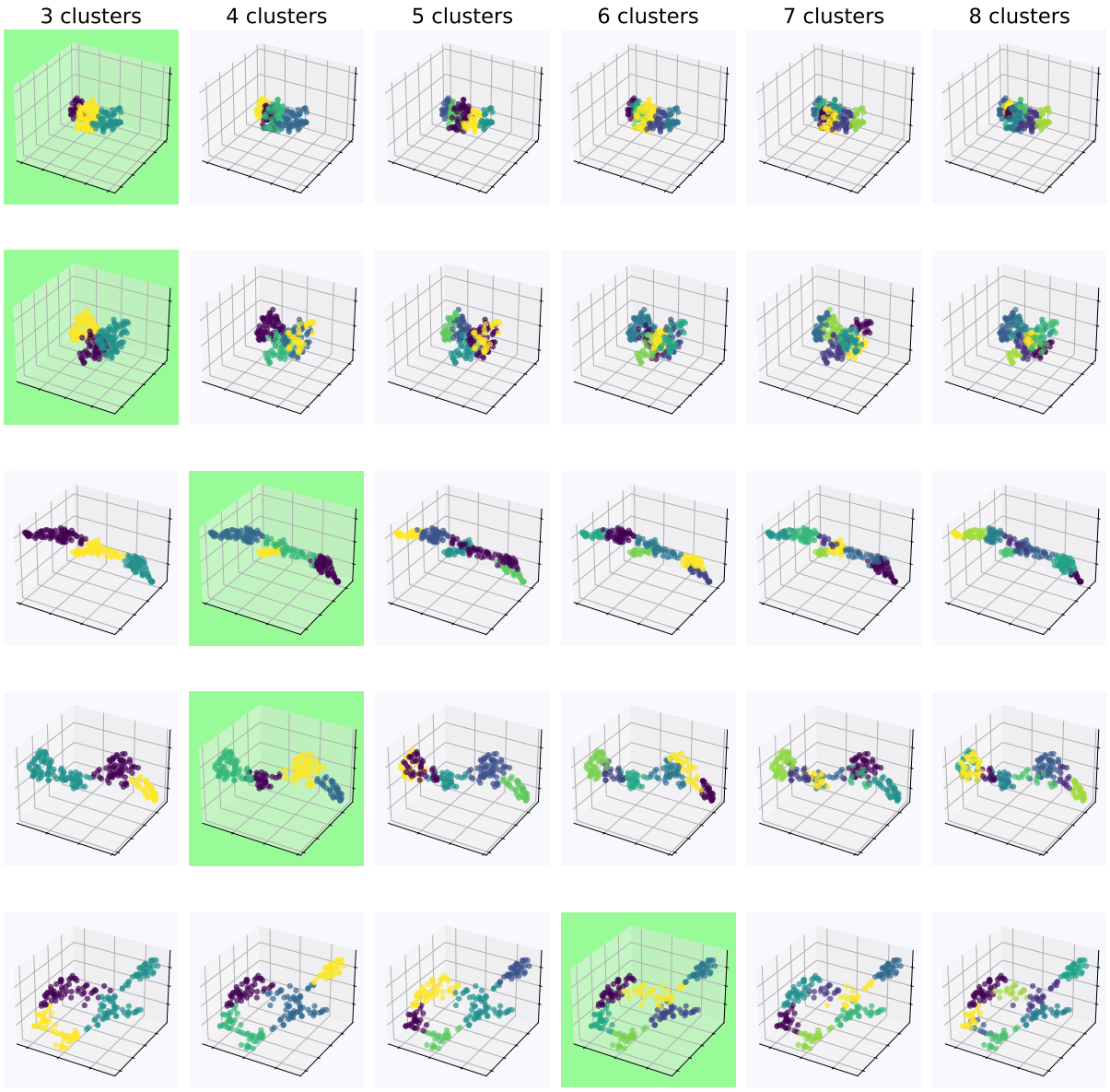

Figure 19: For each dataset we plot the clusters obtained from GMM model, using as input 3 components from **UMAP**. With green color we denote the best GMM model per dataset, with respect to the best BIC score.

3.4 Best Model

The analysis performed using the three dimensionality reduction techniques overall indicates that 2 components seem to suffice for approximating the true distribution of the data. Specifically, both PCA and UMAP seem to give good results for 2 components, however UMAP seem to generate higher quality embeddings in higher dimensions leading to consistent clustering results across all datasets. On the other hand t-SNE seem to yield weak results and cannot be used for generating embeddings in more than 3 dimensions.

Using UMAP and 2 components we can conclude that a good approximation of the true distribution is given using 4 clusters since the first four dataset seem to yield good results for 4 clusters, while the fifth seems to be an outlier that always prefers more clusters in order to explain the underlying data.

Figure 20 presents the posterior distribution of all five datasets using UMAP with 2 components and GMM using a full covariance matrix and 4 states (clusters).

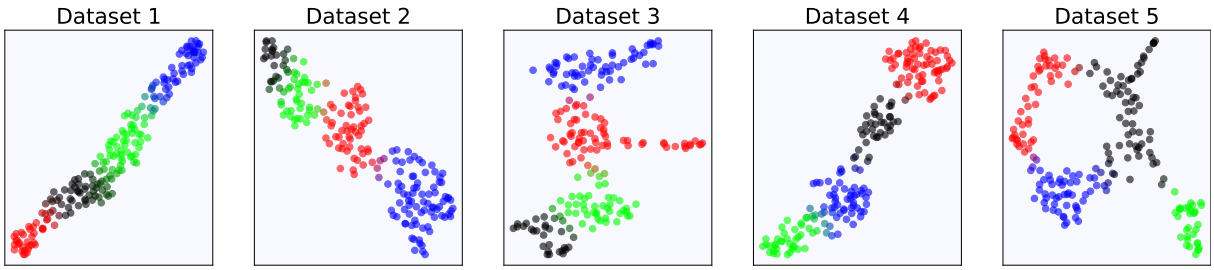

Figure 20: Posterior distribution of the five datasets using UMAP with 2 components and a GMM model having a full covariance matrix and 4 state (clusters).

References

[1] "Appendix E: Model Selection Criterion: AIC and BIC". In: *The Basics of Financial Econometrics: Tools, Concepts, and Asset Management Applications*. John Wiley & Sons, Inc., 2014, pp. 399–403. ISBN: 978-1-118-85640-6 978-1-118-57320-4. DOI: 10.1002/9781118856406.app5.

[2] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[3] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML].

[4] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 03770427. DOI: 10.1016/0377-0427(87)90125-7.

4 Appendix A

This appendix contains information for installing Python dependencies and running the Python notebook submitted along this report.

Installing and Running the Notebook

We chose to perform the data analysis for this assignment using a Jupyter Notebook in Python. We used Python 3.8 during the development of the project. For loading the provided data and perform data manipulations , we employed the the Numpy and Pandas packages that are widely adopted by the community. Moreover, for visualizing the results we employ the MatplotLib package.

In order to perform the dimensionality reduction and clustering tasks, requested by the assignment, we used the SkLearn package that provides a plethora of well documented off-the-shelf machine learning techniques and metrics. The only method that was not provided by the package is Umap, luckily there is a dedicated package for Python written by the inventors the method.

All required dependencies for running the notebook can be installed through PIP by running:

```
pip install numpy pandas matplotlib sklearn umap-learn
```

Runtime Requirements

The notebook does not have any special runtime requirements, apart from specifying the right path for the datasets. This can be set in the second cell of the notebook, using the variable `datasets`.

5 Appendix B

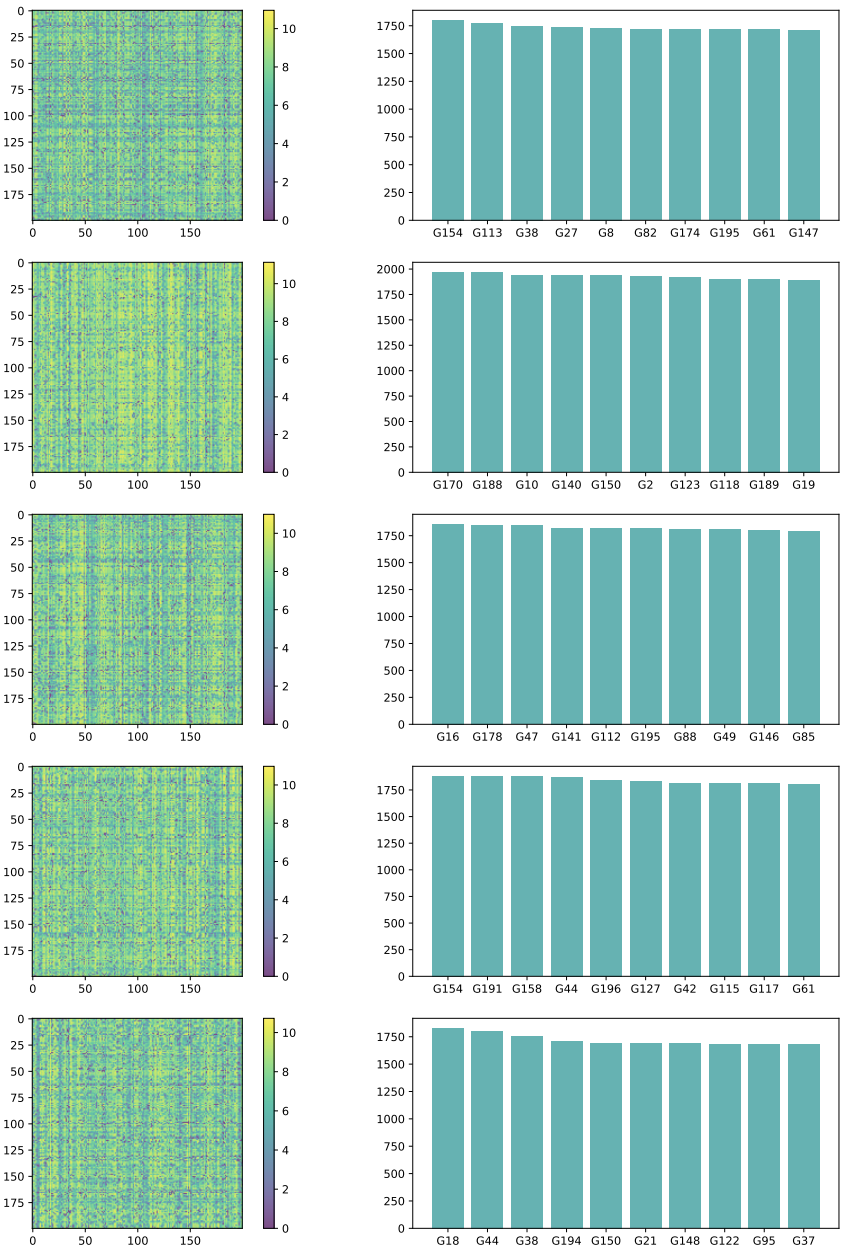

Figure 21: Heatmaps of the genes expressions and the top ten expressed genes.

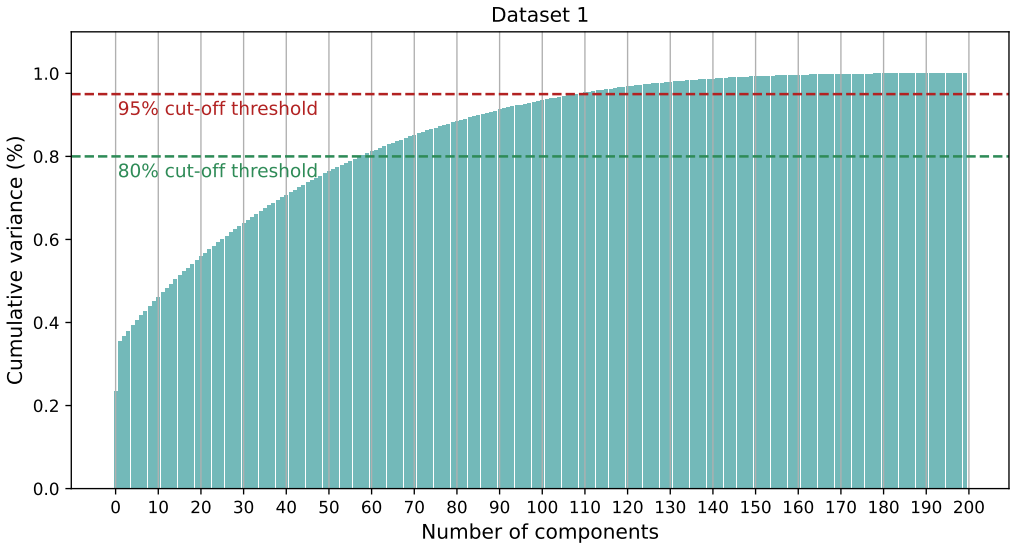

Figure 22: Cumulative Variance explained for Dataset 1.

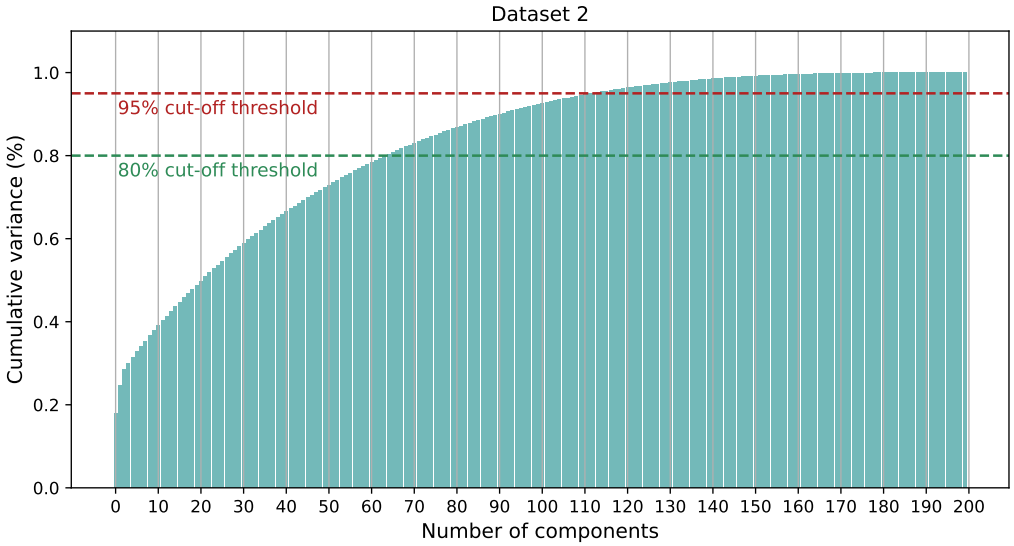

Figure 23: Cumulative Variance explained for Dataset 2.

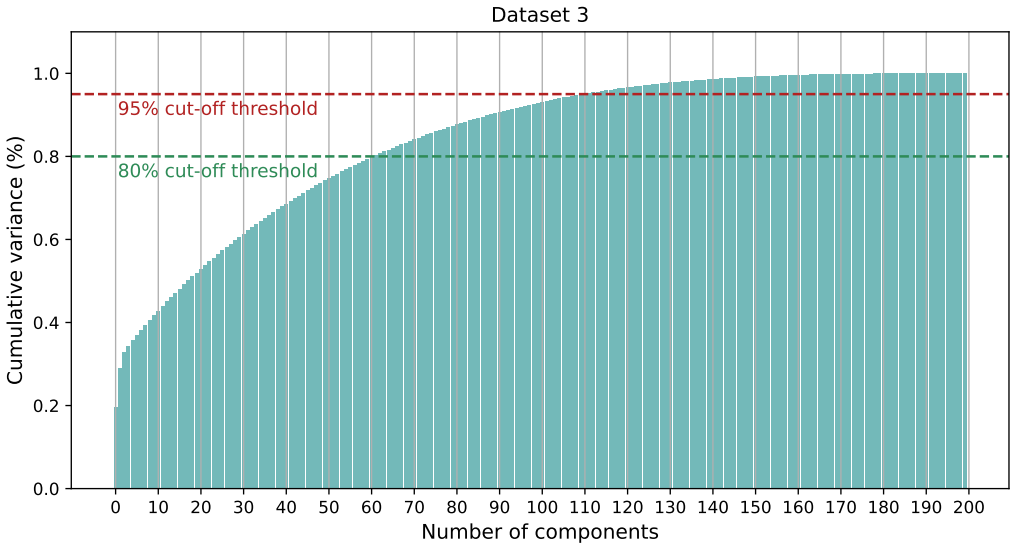

Figure 24: Cumulative Variance explained for Dataset 3.

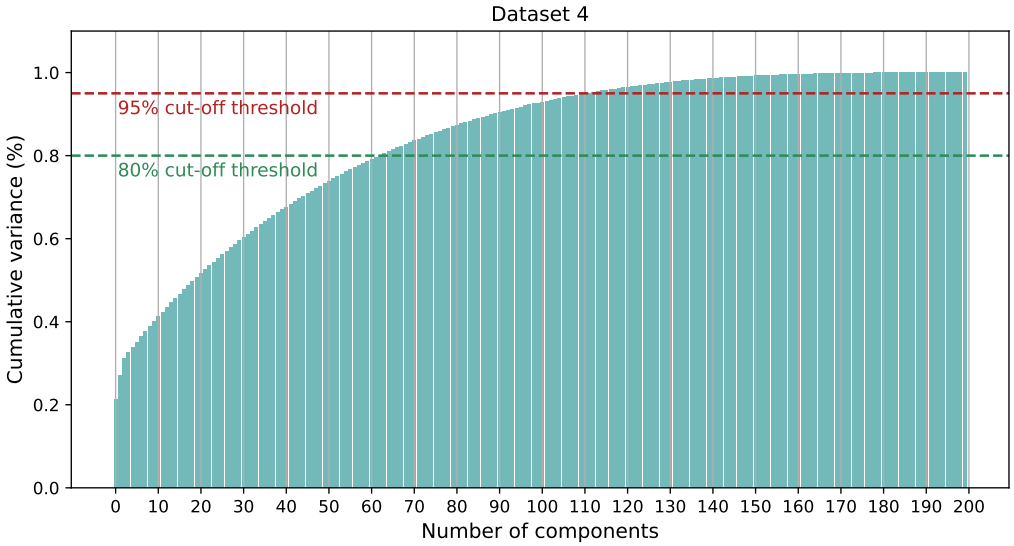

Figure 25: Cumulative Variance explained for Dataset 4.

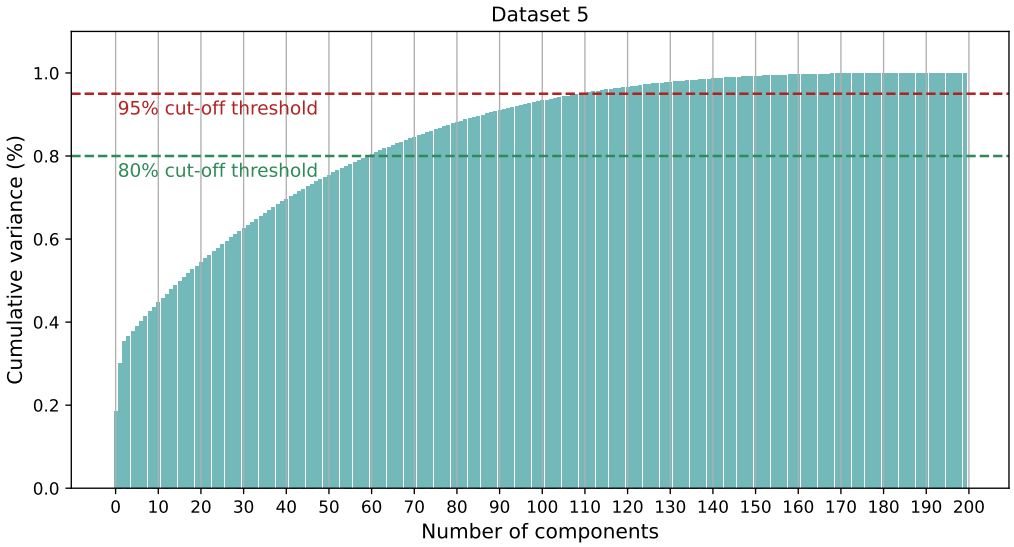

Figure 26: Cumulative Variance explained for Dataset 5.