HELLENIC REPUBLIC
National and Kapodistrian University of Athens

Machine Learning in Computational Biology

<u>2nd Assignment</u>

Dimitra Paranou
7115152100034
Athens, 08/04/2022

Requirements

Python3 was used for the project with the following packages:

1. Pandas, numpy, math, scipy, itertools
2. Sklearn, umap
3. Matplotlib, seaborn, plotly

Dimensionality reduction

Dimensionality reduction is an unsupervised learning technique. It seeks a lower-dimensional representation of numerical input data that preserves the salient relationships in the data. As a result, it captures as much information as the original set of variables. The benefits that we achieve from applying dimensionality reduction to a dataset are:

- Less dimensions that leads to less computation/training time
- Less required space to store the data
- As some algorithms do not perform well when we have large dimensions, through this technique, these algorithms become more useful
- Takes care of multicollinearity by removing redundant features
- Helps in data visualization

Following, are presented some dimensionality reduction techniques that were used in our datasets.

Before that, we have to declare that the columns in Datasets are the Cells and the rows are the Genes. Also, Standardization of features by removing the mean and scaling to unit variance was performed.

Lastly, for each dataset we have to choose the parameters that are ideal to it, but in this code we choose more general parameters for datasets (parameters that fit to most of the datasets so as to present all datasets).

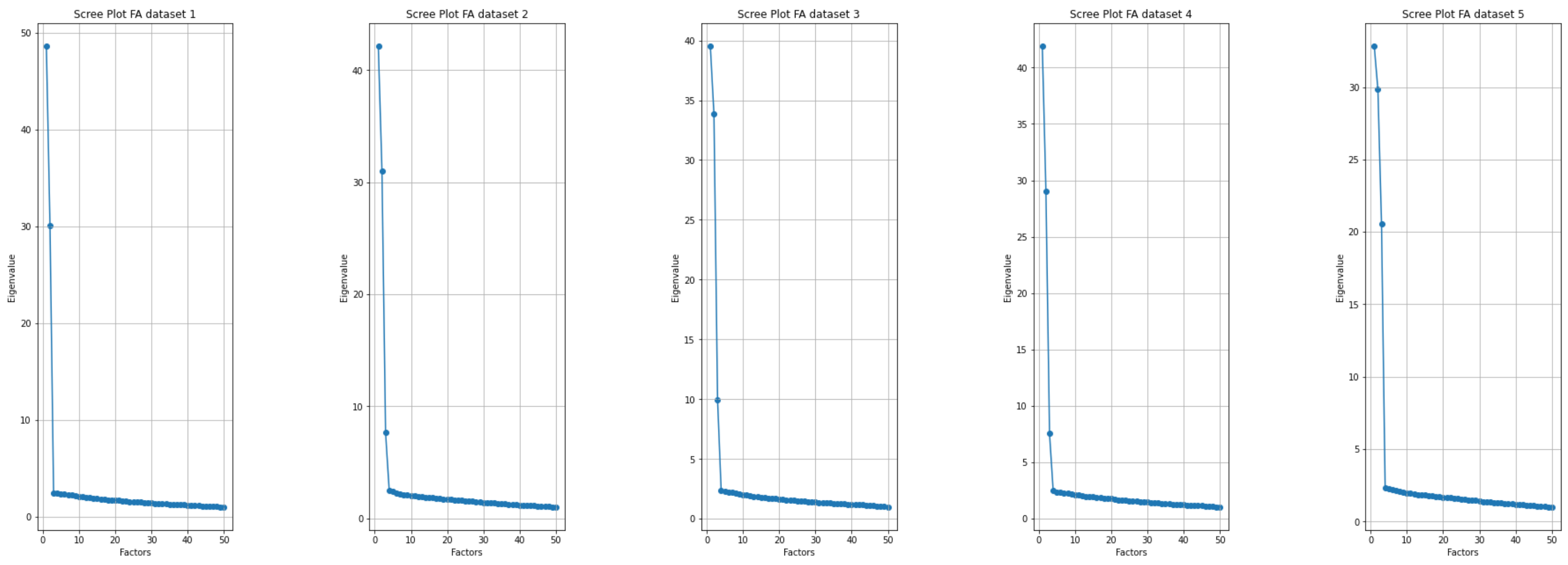
Factor Analysis

Factor Analysis technique, variables are grouped by their correlations, i.e., all variables in a particular group will have a high correlation among themselves, but a low correlation with variables of other group(s). It is a simple linear generative model with Gaussian latent variables. Here, each group is known as a factor. These factors are small in number as compared to the original dimensions of the data. However, these factors are difficult to observe.

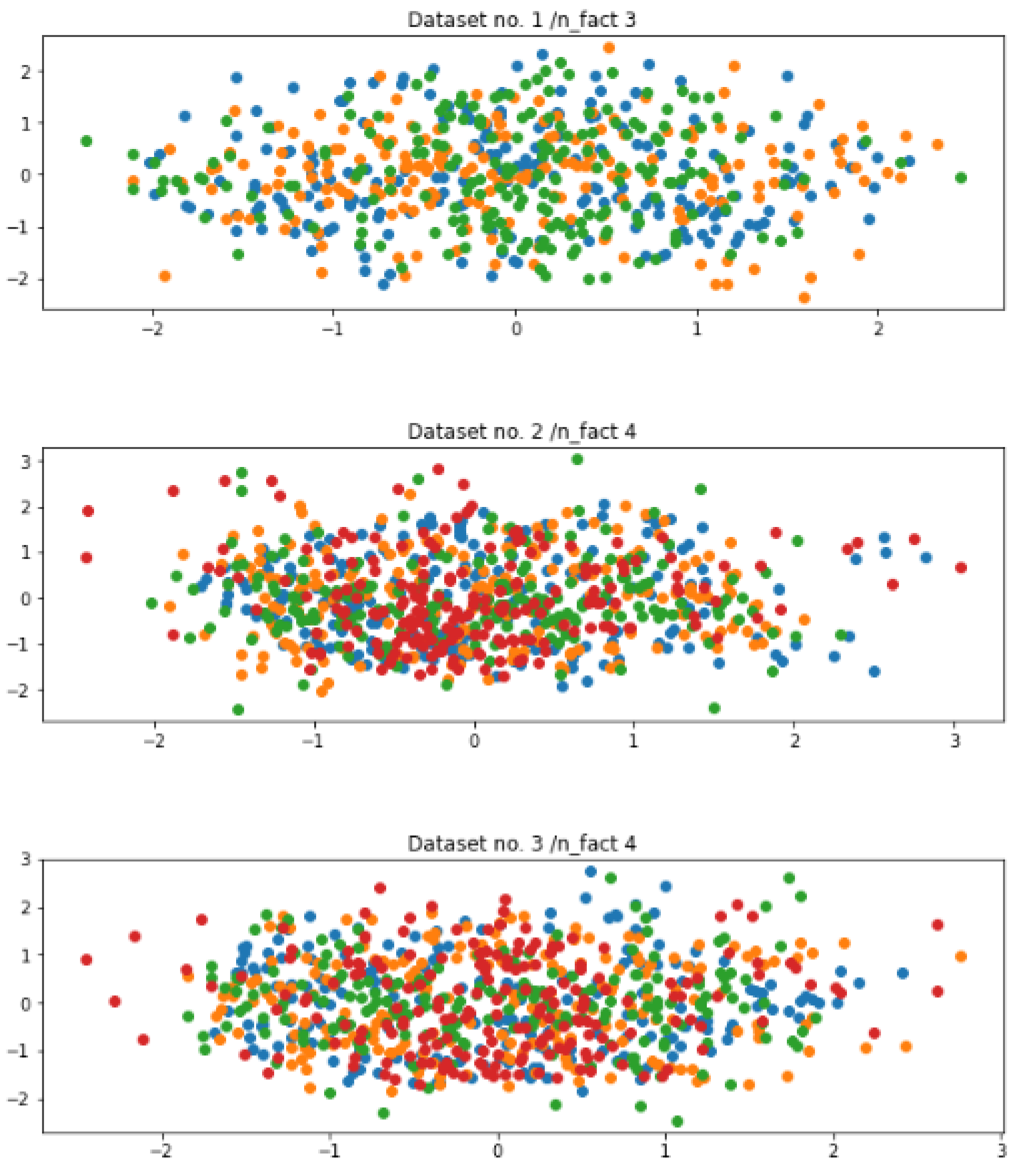
FactorAnalysis performs a maximum likelihood estimate of the so-called loading matrix, the transformation of the latent variables to the observed ones, using SVD based approach.

To find the optimal number of factors, FactorAnalyzer (package factor_analyzer) is used without any parameters, so as to identify the optimal number of factors. The eigenvalues that are greater than one, are considered as the number of factors. In the datasets, the number of factors are presented below and a screeplot from the last dataset is also given.

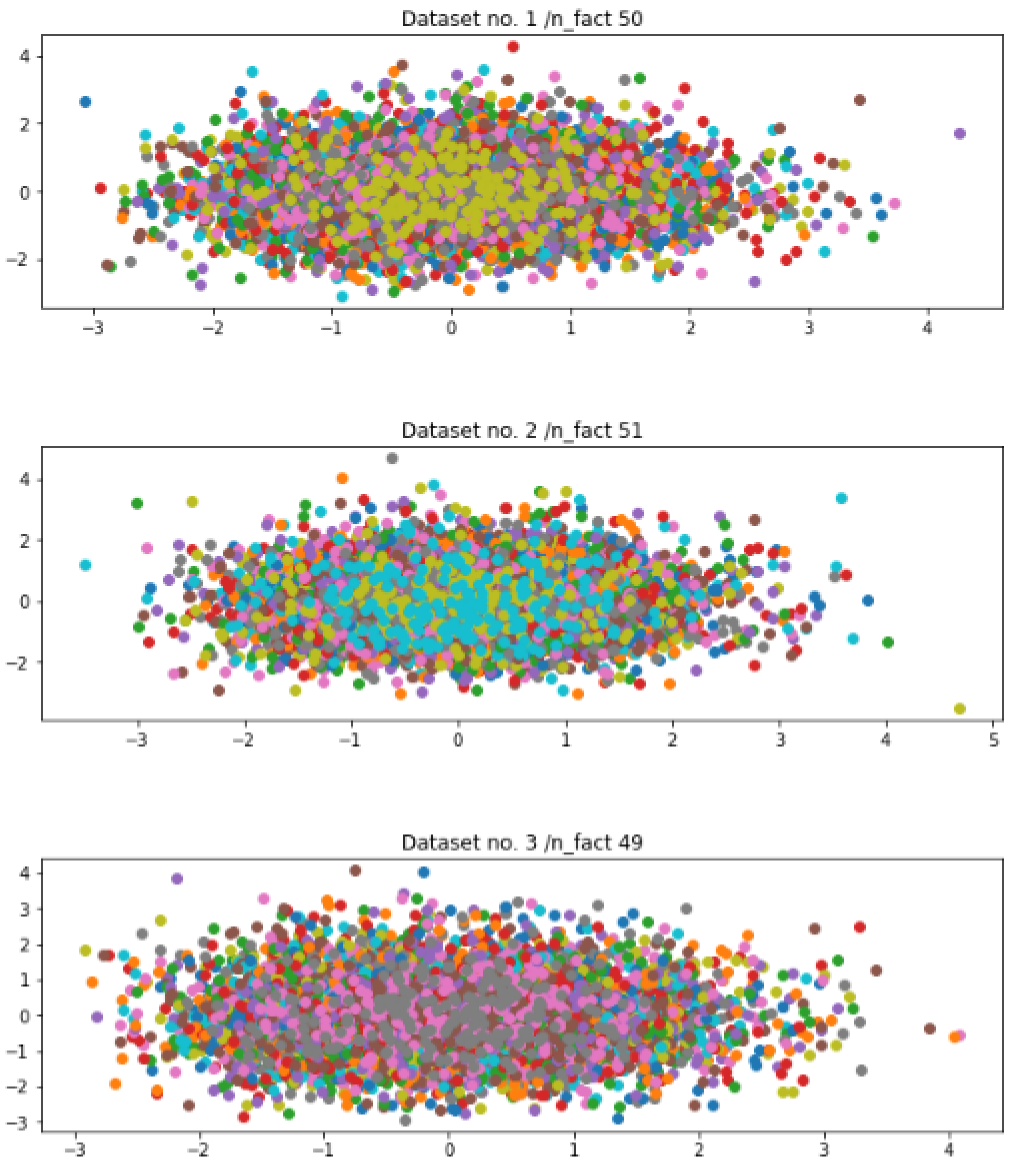
	Number of Factors
Dataset 1	50
Dataset 2	51
Dataset 3	49
Dataset 4	52
Dataset 5	50



The usage of Factor Analysis (package sklearn) requires only the number of components (that we identified before). The following plots that are produced from the datasets are given below. We kept only the factors that contain the most information (ex. Dataset1 - 3, Dataset2 - 4, etc.) in the first screenshot and all factors in the second screenshot (just for illustration) (first 3 datasets)



FA with the most important factors



FA with all the factors

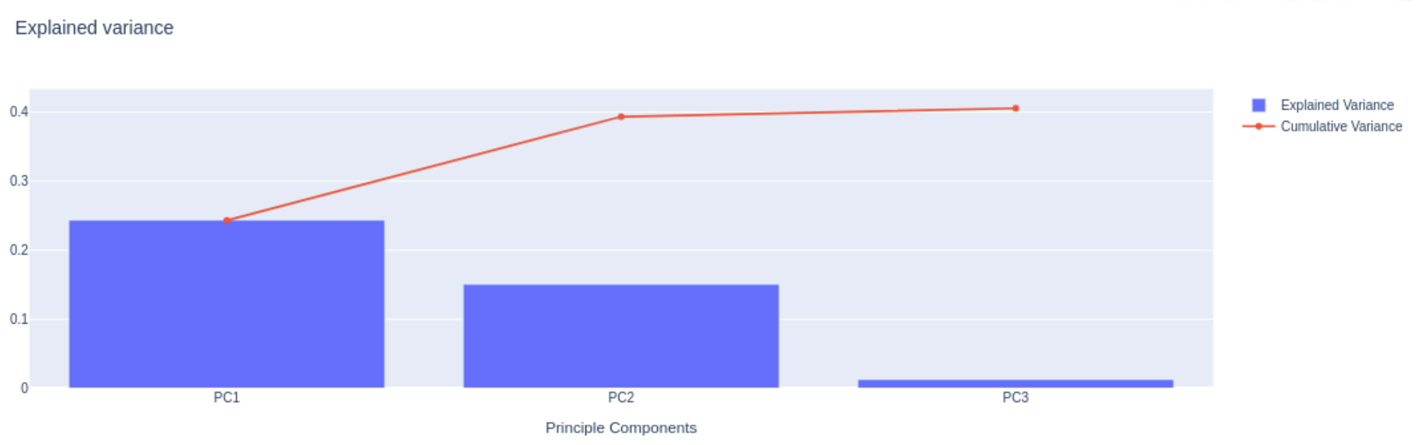
Principal Component Analysis (PCA)

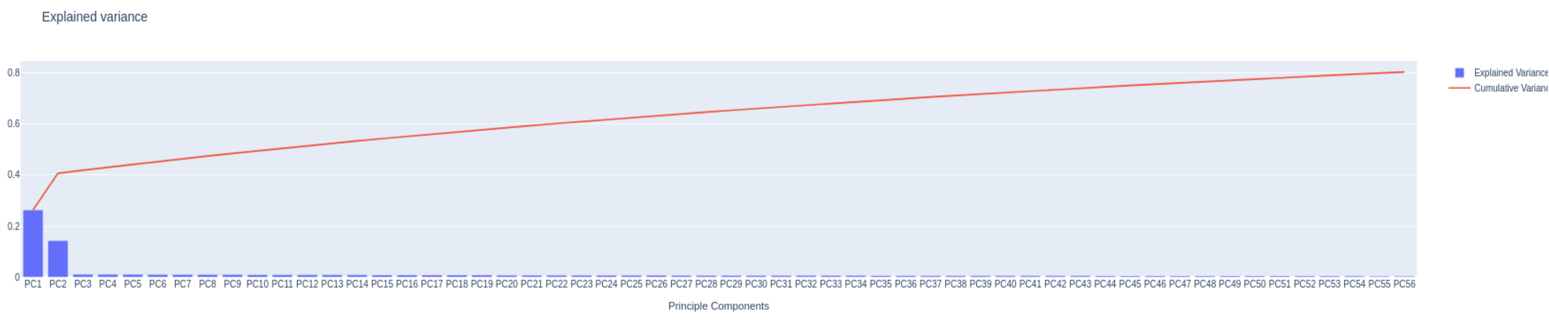
PCA is a technique which helps in extracting a new set of variables from an existing large set of variables. It extracts low dimensional set of features by taking a projection of irrelevant dimensions from a high dimensional data set with a motive to capture as much information as possible.

A principal component is a normalized linear combination of the original predictors in a data set. The first principal component is a linear combination of original predictor variables which captures the maximum variance in the data set. It determines the direction of highest variability in the data. Larger the variability captured in the first component, larger the information captured by component. No other component can have variability higher than the first principal component.

The principal components are supplied with normalized version of original predictors. Performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance.

The usage of PCA (package sklearn) requires only the number of components, where it is set to 3. In the first dataset, we need 56 components for 80% of the information for example.

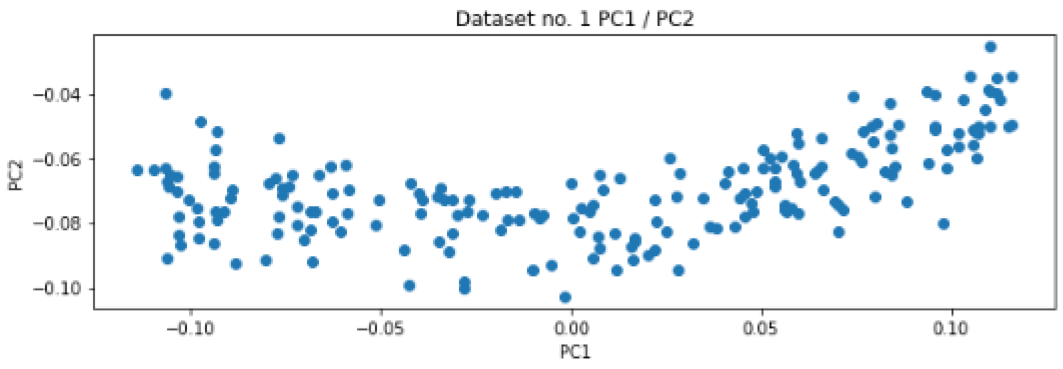


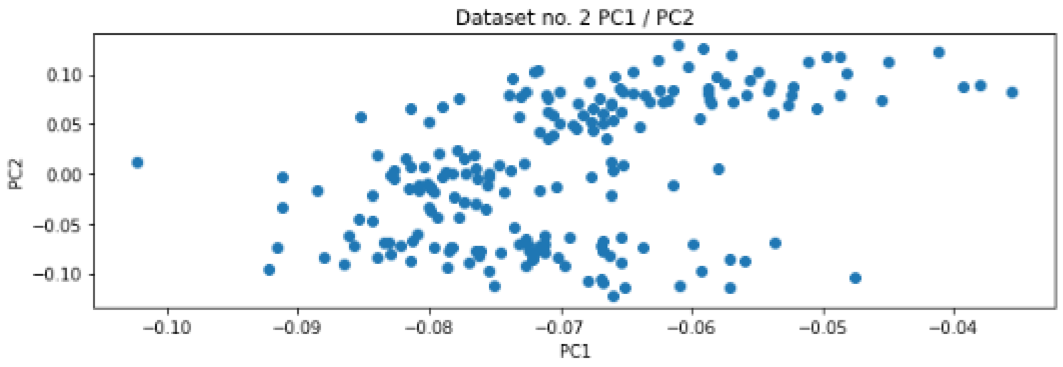


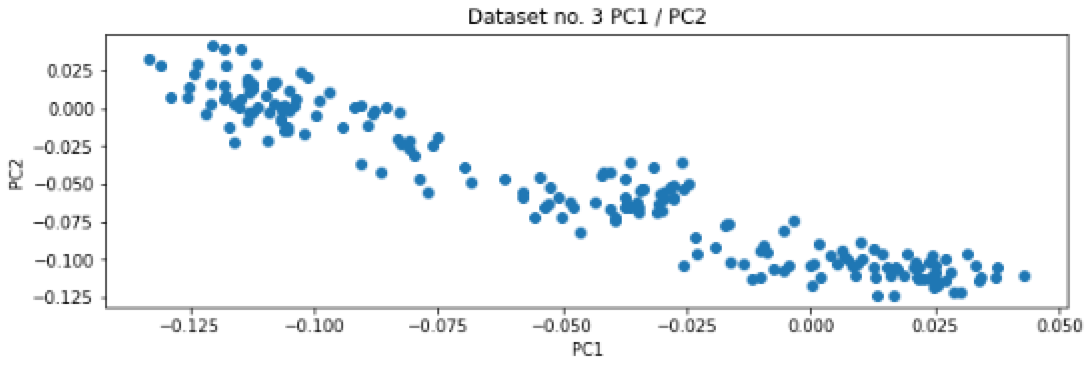
For all the datasets, the desire number of PC is presented below:

	Number of Components
Dataset 1	56
Dataset 2	56
Dataset 3	55
Dataset 4	57
Dataset 5	56

Then we plot the first and the second PC for visualization of the datasets. (first 3 datasets)







The total variance for each dataset for 2 PC is given below:

	Variance
Dataset 1	0.41
Dataset 2	0.36
Dataset 3	0.37
Dataset 4	0.36
Dataset 5	0.32

For PCA, we keep only the first and the second component as they give more information. The rest of the components do not contribute so much to the information so we do not want them.

Singular Value Decomposition (SVD)

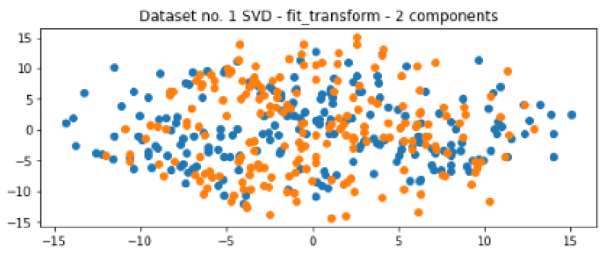
This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently.

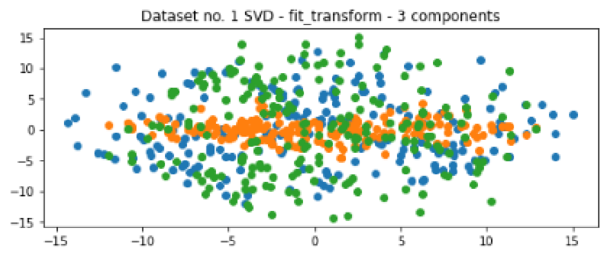
SVD decomposes the original variables into three constituent matrices. It is essentially used to remove redundant features from the dataset. It uses the concept of Eigenvalues and Eigenvectors to determine those three matrices.

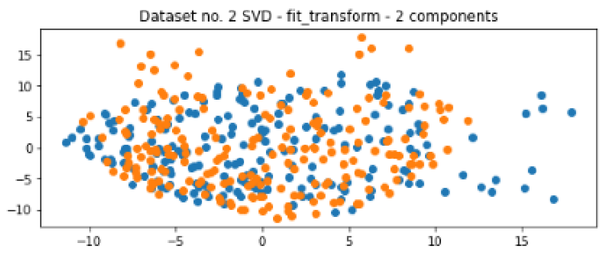
The usage of SVD (package sklearn) requires the following parameters:

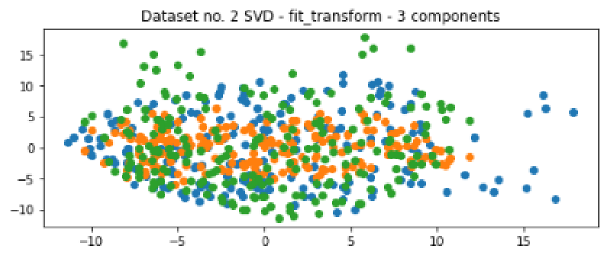
- Number of components
- Random state : Used during randomized svd. Pass an int for reproducible results across multiple function calls

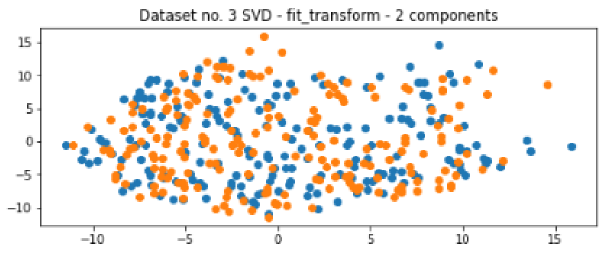
In the datasets, we use the fit method and the fit_tranform method, where the latter fits the model to the data and performs dimensionality reduction on the data. In both cases we select 2 and 3 components, but the latter represent a better distinction of the components. Following are the 2 relative plots. (first 3 datasets)

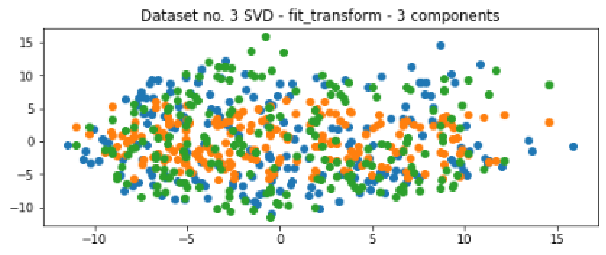


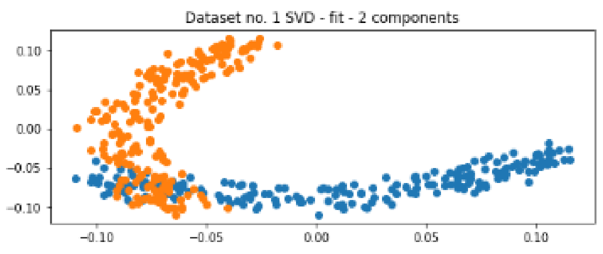


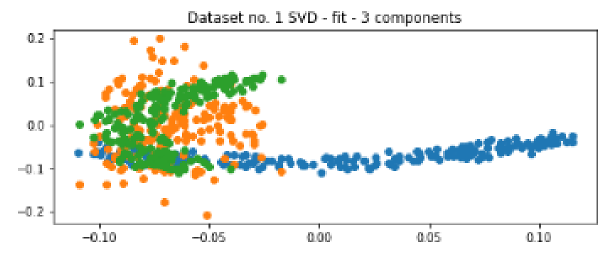


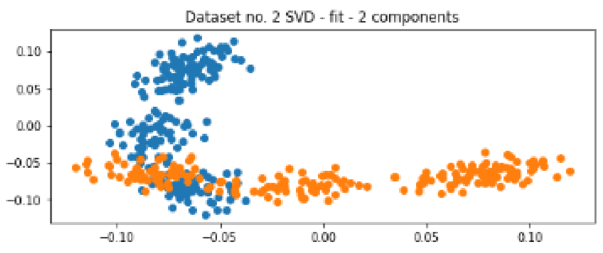


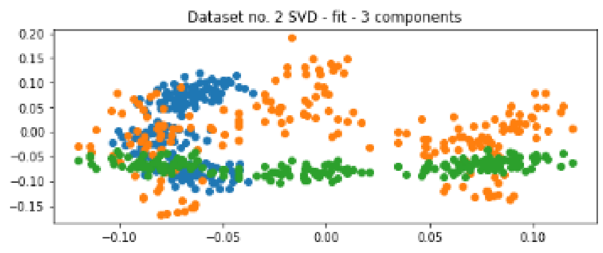


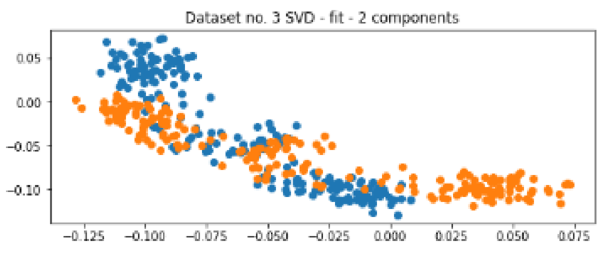


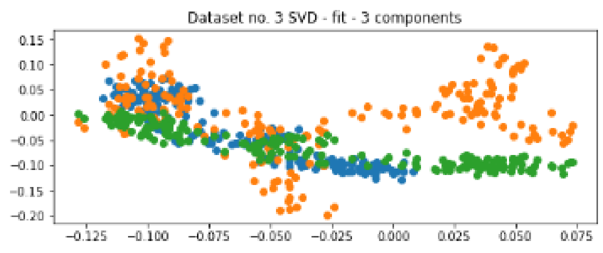












In this case the total variance for each dataset is:

	Variance for 2 Components / 3 Components
Dataset 1	0.39 / 0.41
Dataset 2	0.37 / 0.4
Dataset 3	0.37 / 0.42
Dataset 4	0.35 / 0.39
Dataset 5	0.31 / 0.42

T-distributed stochastic neighbor embedding (tsne)

t-SNE is a more advance technique for easily searching patterns in a non-linear way. It is one of the few algorithms which is capable of retaining both local and global structure of the data at the same time and it calculates the probability similarity of points in high dimensional space as well as in low dimensional space.

High-dimensional Euclidean distances between data points are converted into conditional probabilities that represent similarities using the formula:

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

where xi and xj are data points, ||xi-xj|| represents the Euclidean distance between these data points, and **σ**i is the variance of data points in high dimensional space.

For the low-dimensional data points yi and yj corresponding to the high-dimensional data points xi and xj, it is possible to compute a similar conditional probability using:

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i}\exp\left(-\|y_i - y_k\|^2\right)}$$

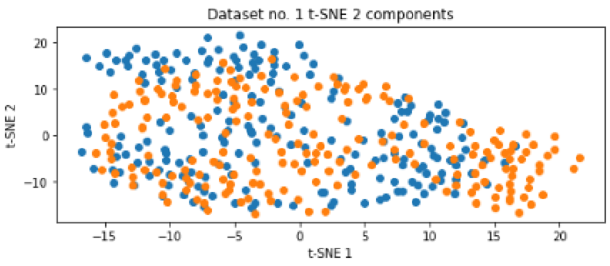
where ||yi-yj|| represents the Euclidean distance between yi and yj.

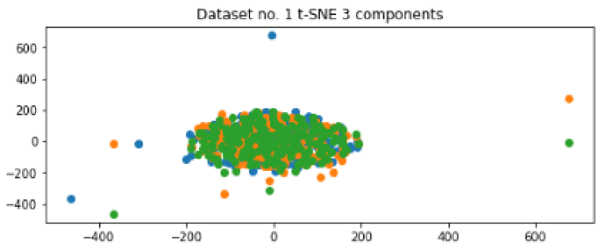
After calculating both the probabilities, it minimizes the difference between both the probabilities

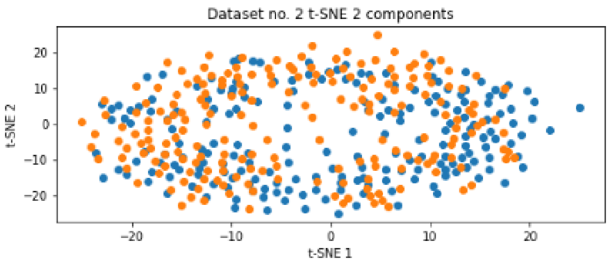
The usage of TSNE (package sklearn) requires the following parameters:

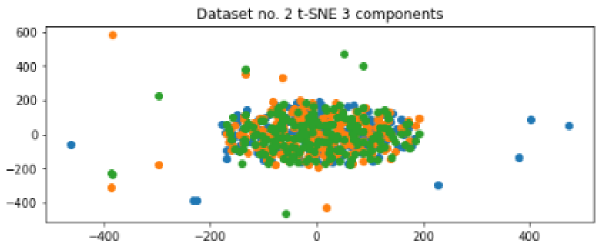
- Number of components (no more than 3)
- Perplexity: which is related to the number of nearest neighbors that is used in other manifold learning algorithms (we get the square of the dataset length - power law obtained from experiment)
- Number of iterations that the method will run

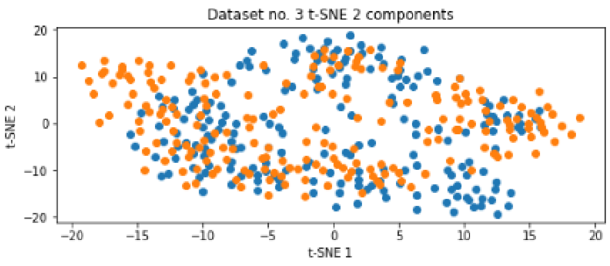
Following is the plot of 2 and 3 components of the datasets. (first 3 datasets)

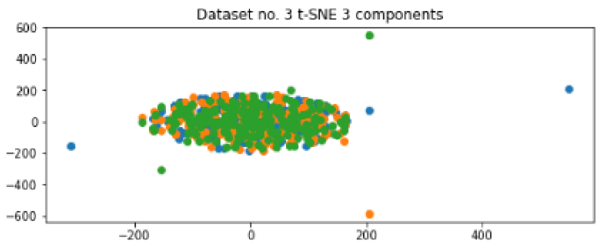












Even if t-SNE works very well on large datasets, it also has its limitations, such as loss of large-scale information, slow computation time, and inability to meaningfully represent very large datasets.

UMAP

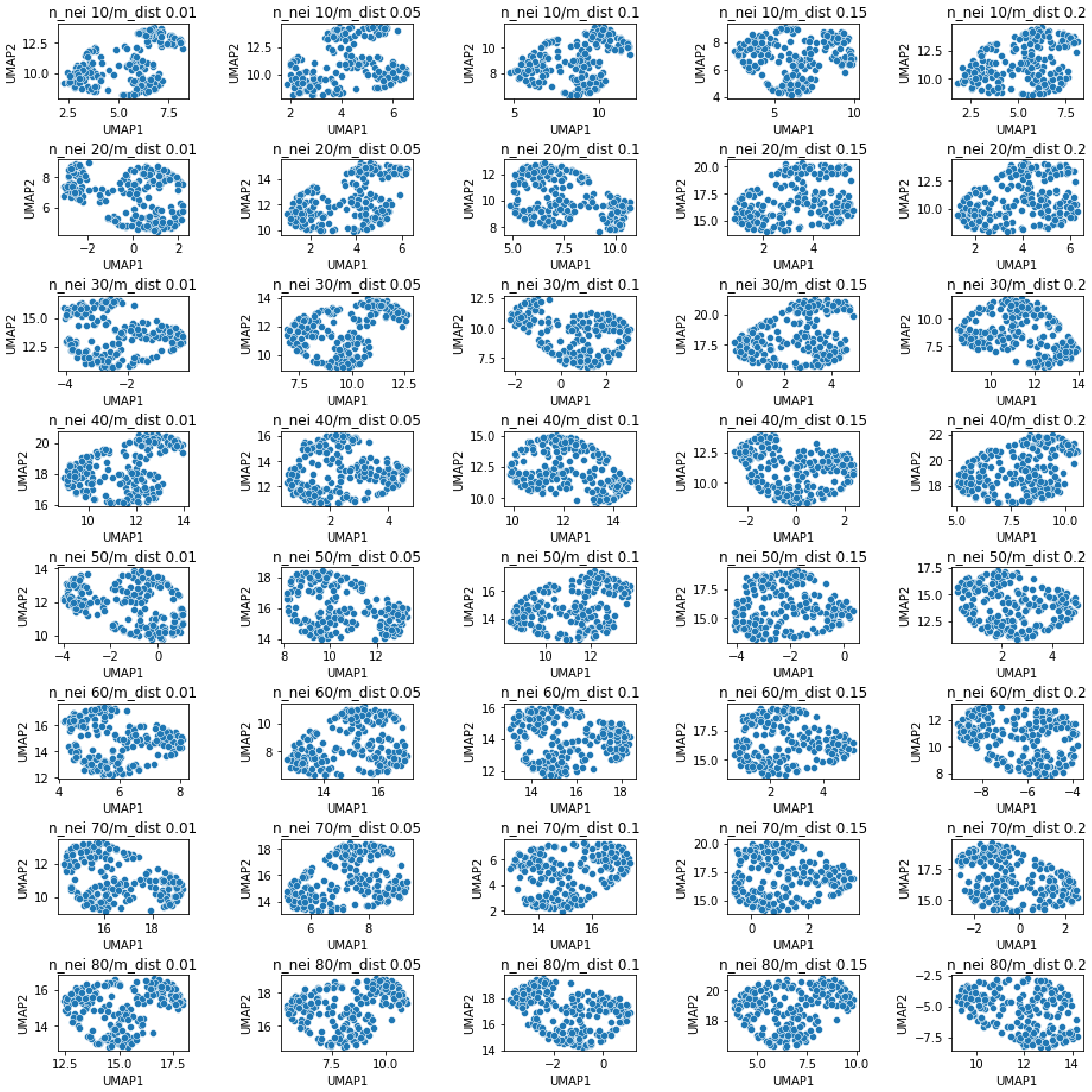
Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can preserve as much of the local, and more of the global data structure as compared to t-SNE, with a shorter runtime. It can handle large datasets and high dimensional data without too much difficulty, combines the power of visualization with the ability to reduce the dimensions of the data and along with preserving the local structure, it also preserves the global structure of the data.

This method uses the concept of k-nearest neighbor and optimizes the results using stochastic gradient descent. It first calculates the distance between the points in high dimensional space, projects them onto the low dimensional space, and calculates the distance between points in this low dimensional space. It then uses Stochastic Gradient Descent to minimize the difference between these distances.

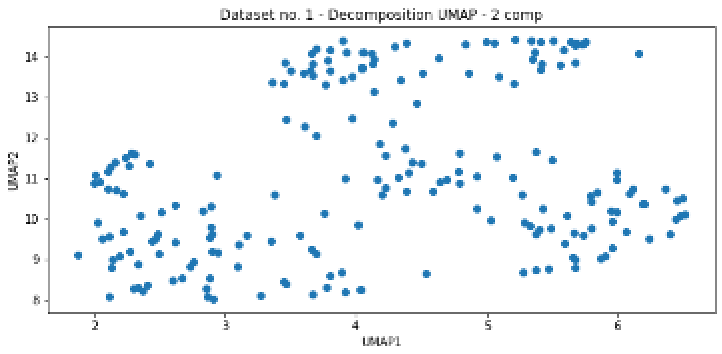
The usage of UMAP (package umap) requires the following parameters:

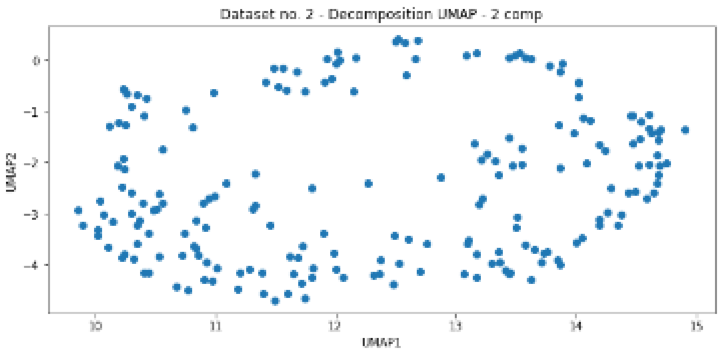
- Number of neighbors (of points)
- Minimum distance: which controls how tightly embedding is allowed
- Random state

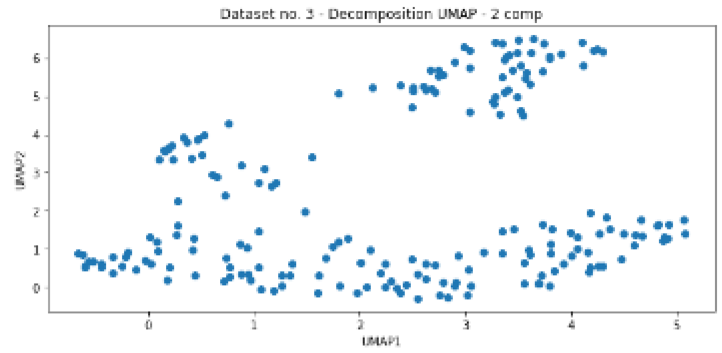
To determine the various effects of the parameters, we try in the first dataset the UMAP and following are the results of the several tries.



As number of neighbors and minimum distance are the keys for UMAP, we select n_neighbors = 10 (see more of the overall structure of the data, gluing more components together, and better covering the broader structure of the data) and min_dist = 0.05 (better overarching view of the data at the loss of the more detailed topological structure). The following plots are the result. (first 3 datasets)







In addition, we applied first PCA to our dataset with a number of components that achieves 80% of variance and then we applied UMAP with the appropriate parameters. Due to the fact that the results were almost the same as applying UMAP in the original data, we kept the first approach. However, it's a widely used technique that can also be applied.

Having performed many dimensionality reduction techniques, we can see that each one has its own characteristics and is appropriate for several situations (some of them are more flexible and can be used in more datasets than others). In our dataset, SVD and PCA need many components to achieve high variance (the higher the variance the more information we get) as well as Factor analysis needs many factors. On the other hand, for t-sne and UMAP we can see that the correlation between the components obtained from UMAP is quite less as compared to the correlation between the components obtained from t-SNE. Hence, UMAP tends to give better results and we will continue our clustering using UMAP results.

Clustering

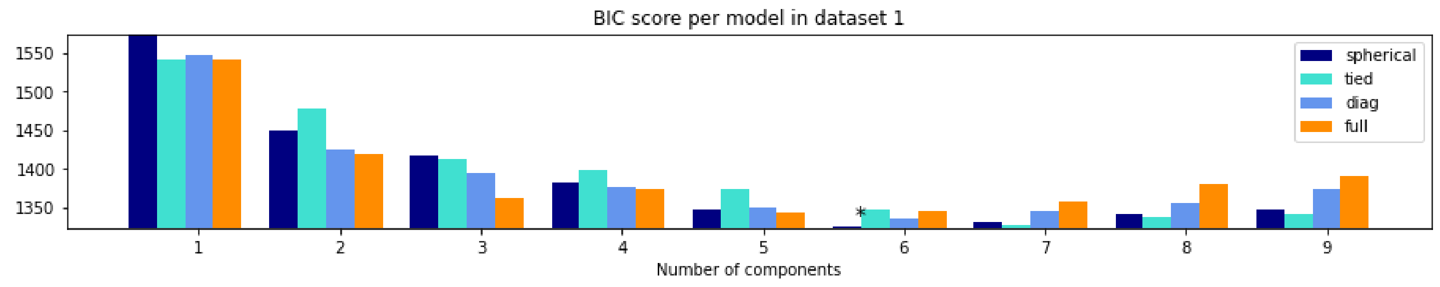
Gaussian Mixture Modeling

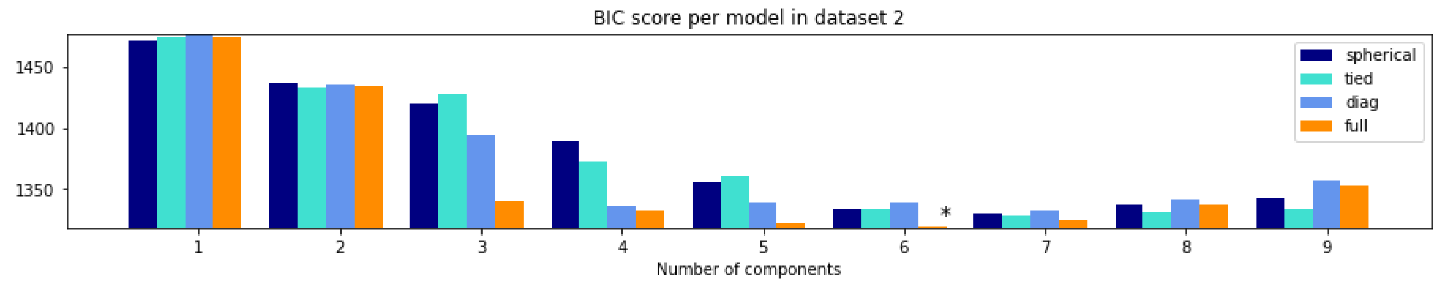
A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. It implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models.

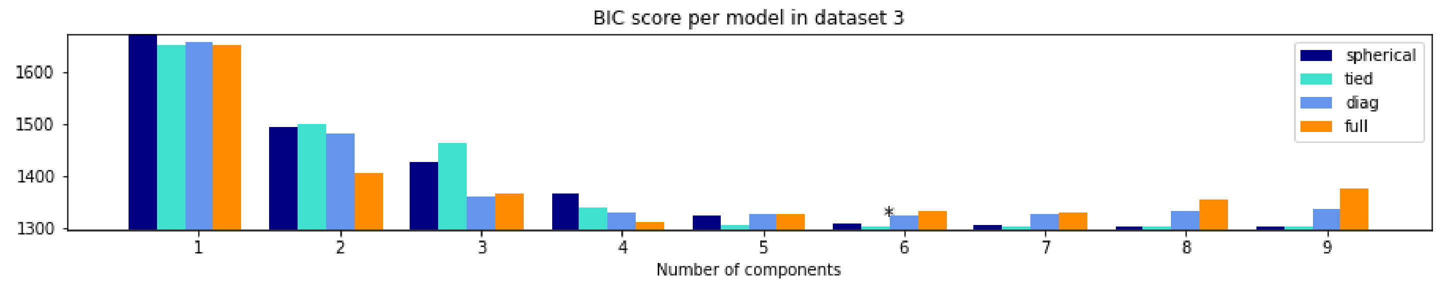
The usage of Gaussian mixture model (package sklearn) requires 2 parameters.

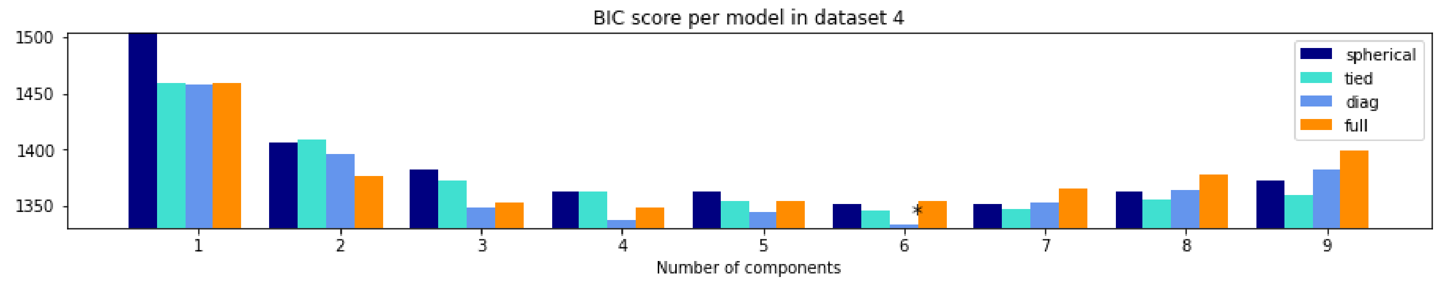
- Number of components
- Covariance type
    - full: each component has its own general covariance matrix
    - tied: all components share the same general covariance matrix.
    - diag: each component has its own diagonal covariance matrix.
    - spherical: each component has its own single variance.

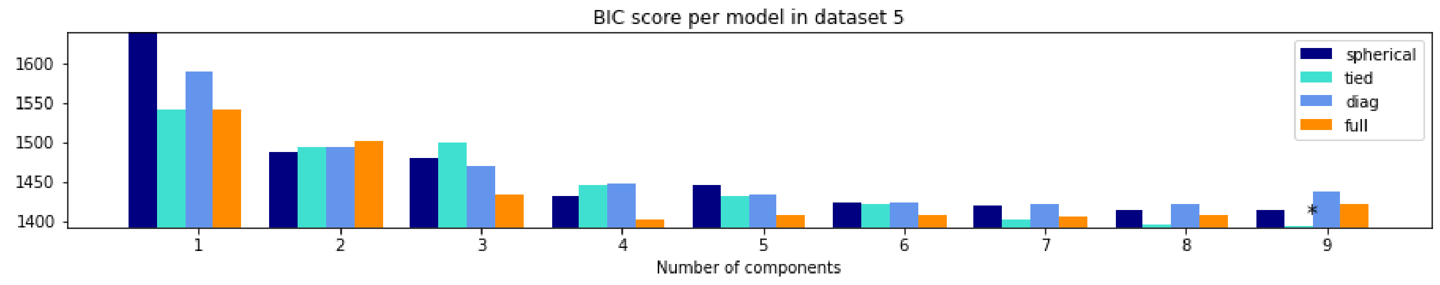
For selecting the number of components in a classical Gaussian Mixture Model and the covariance matrix structure, the BIC criterion was used. (with * is indicated the best model)



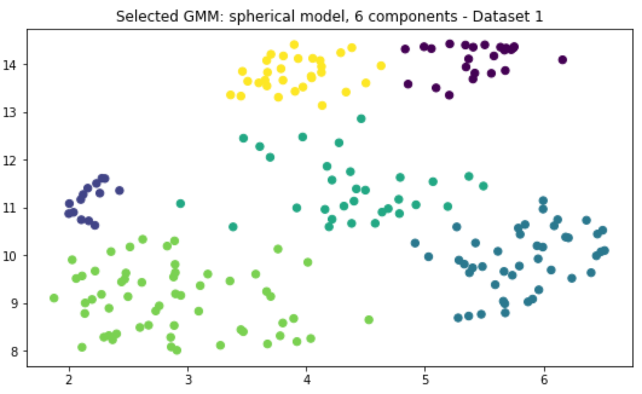


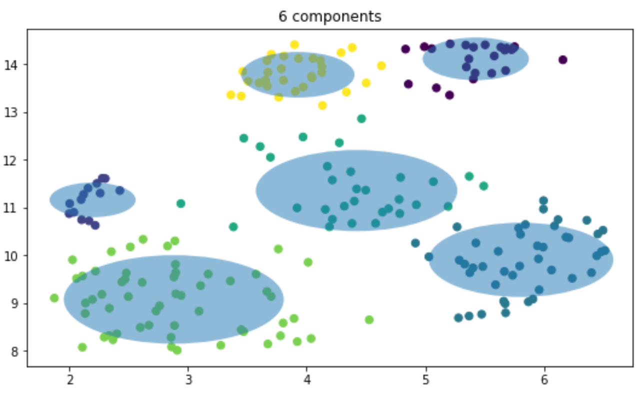


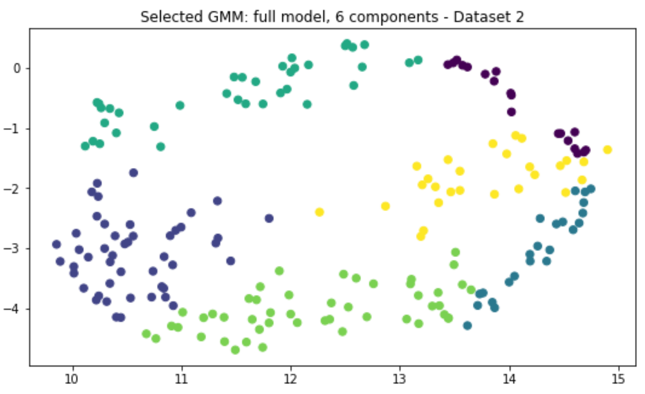


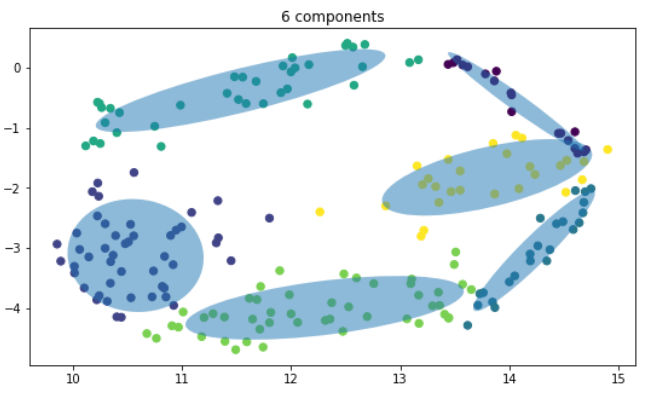


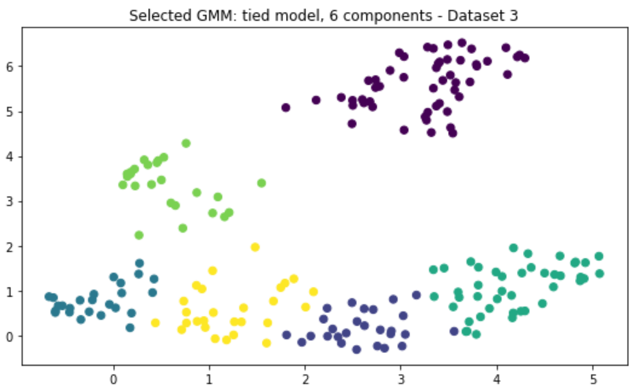
Based on BIC, for the first dataset, the optimal number of Components is 6 and the covariance type is spherical. Given that, we separate the points into the appropriate clusters. The ellipsoids are constructed based on the covariance matrices of each dataset and the following figure represents the datasets and the relative clusters.

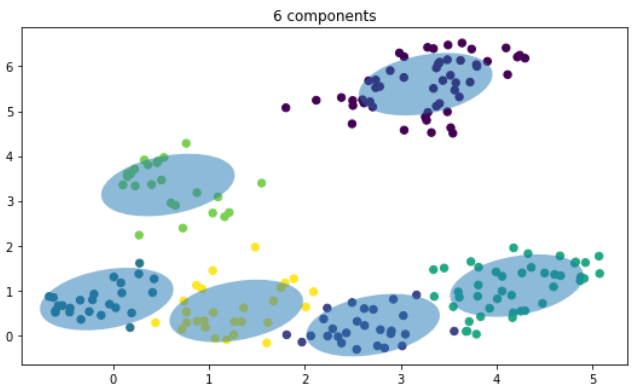

Selected GMM: spherical model, 6 components - Dataset 1

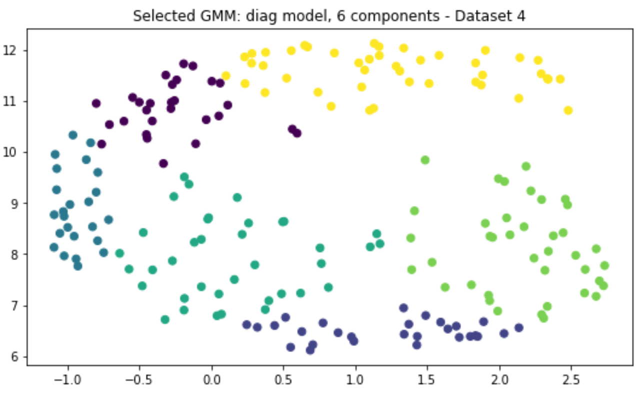

6 components

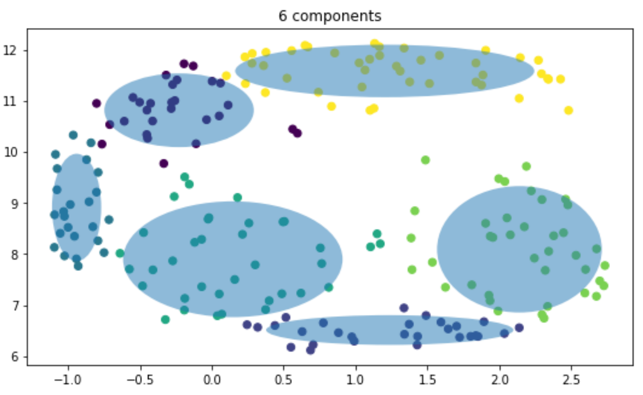

Selected GMM: full model, 6 components - Dataset 2

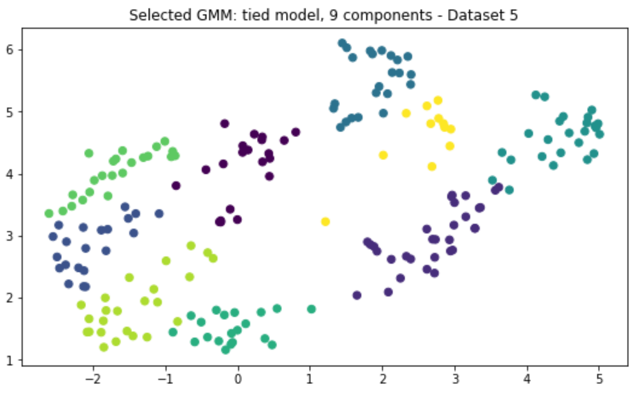

6 components

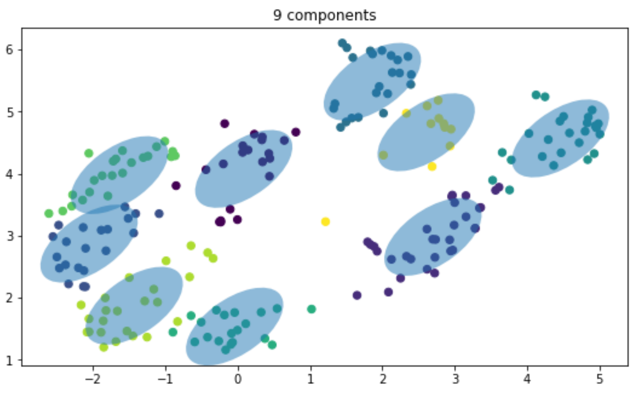

Selected GMM: tied model, 6 components - Dataset 3


6 components

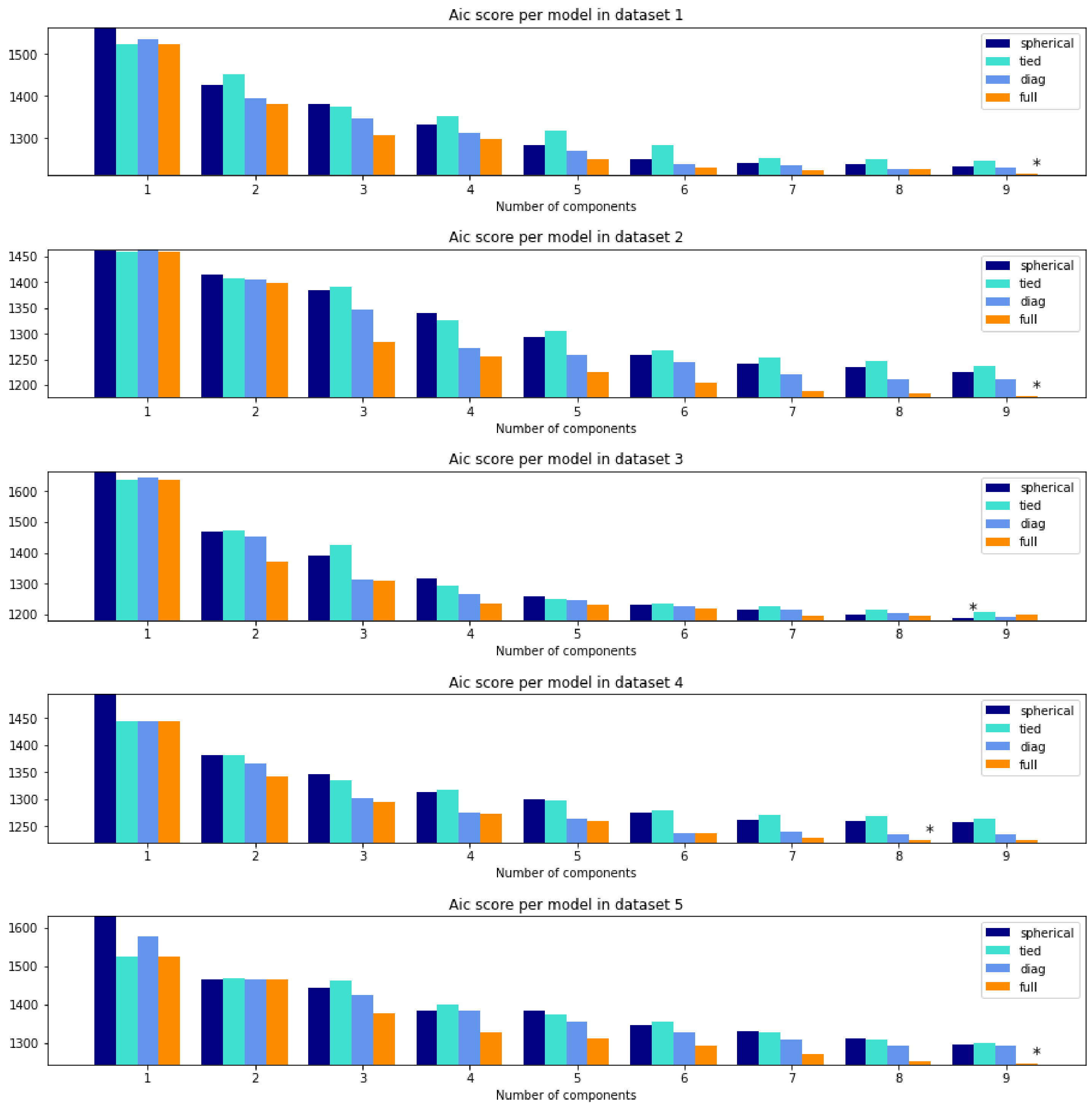

Selected GMM: diag model, 6 components - Dataset 4


6 components

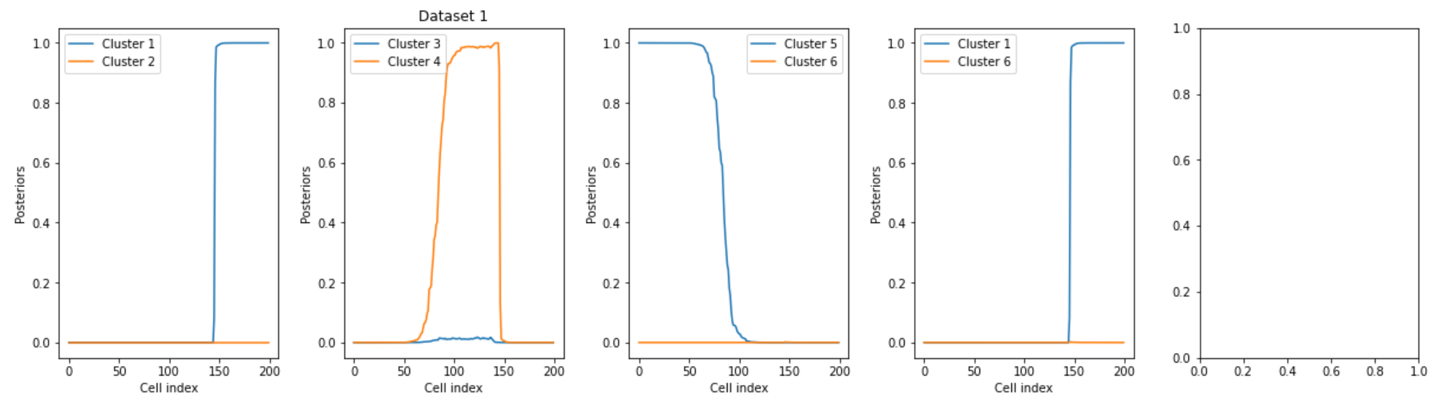

Selected GMM: tied model, 9 components - Dataset 5

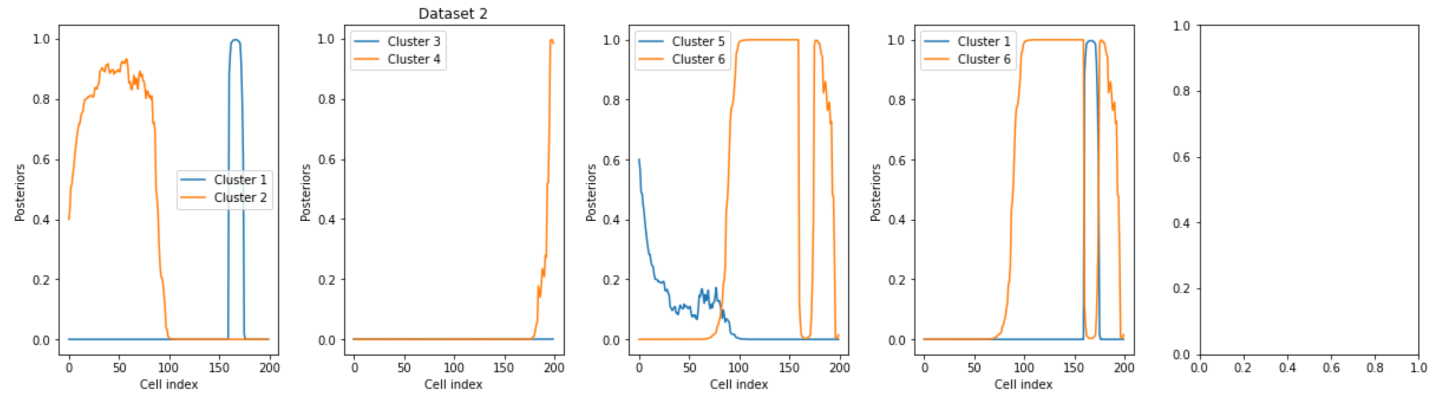

9 components

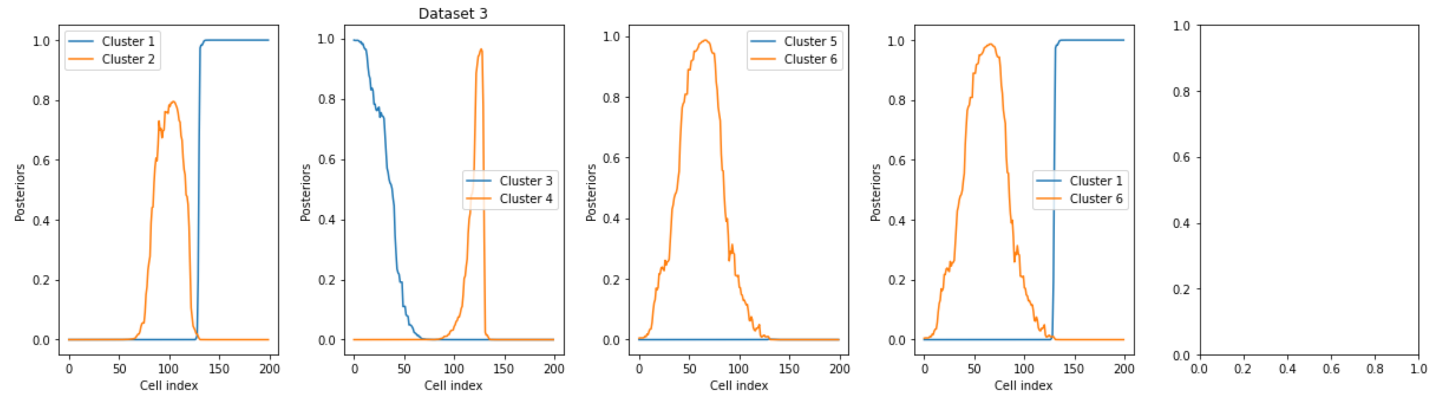
Also the AIC (Akaike information criterion) was calculated for the models and the components. This metric shows that the optimal number of Components is 8 and 9 and the covariance type is full in almost all the datasets.

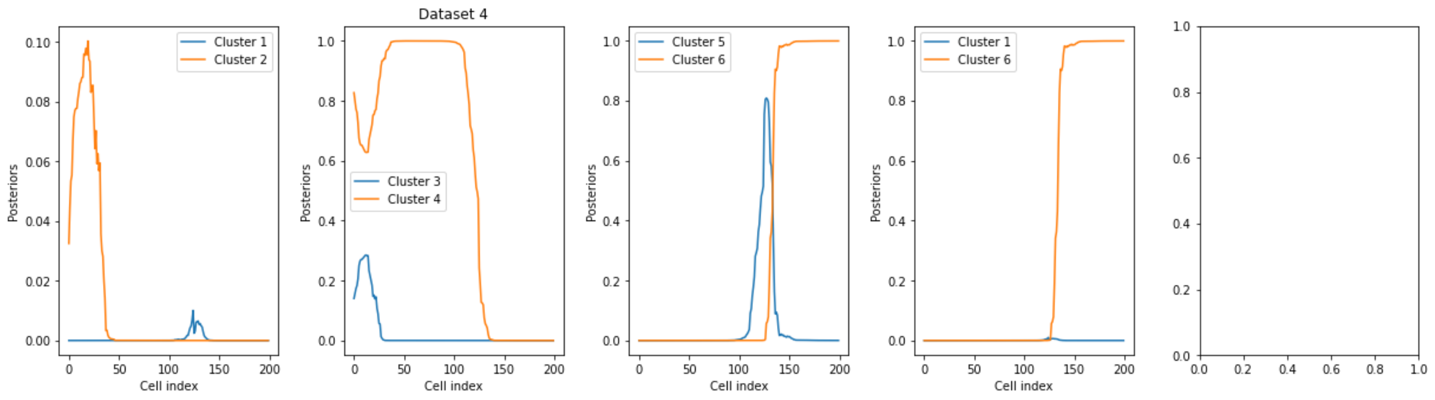


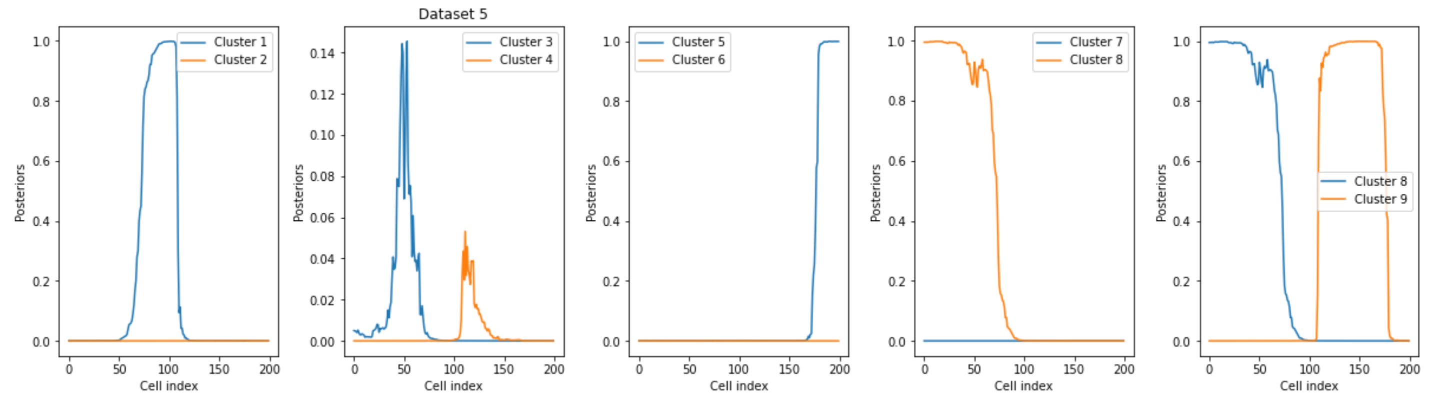
Lastly, we predict posterior probability of data under each Gaussian in the model of datasets and we plot the relative posteriors.


Dataset 1
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 1
Cluster 6
Posteriors
Cell index

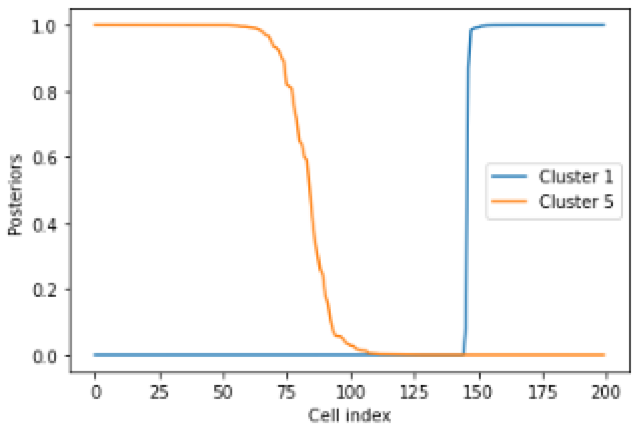

Dataset 2
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 1
Cluster 6
Posteriors
Cell index


Dataset 3
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 1
Cluster 6
Posteriors
Cell index


Dataset 4
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 1
Cluster 6
Posteriors
Cell index

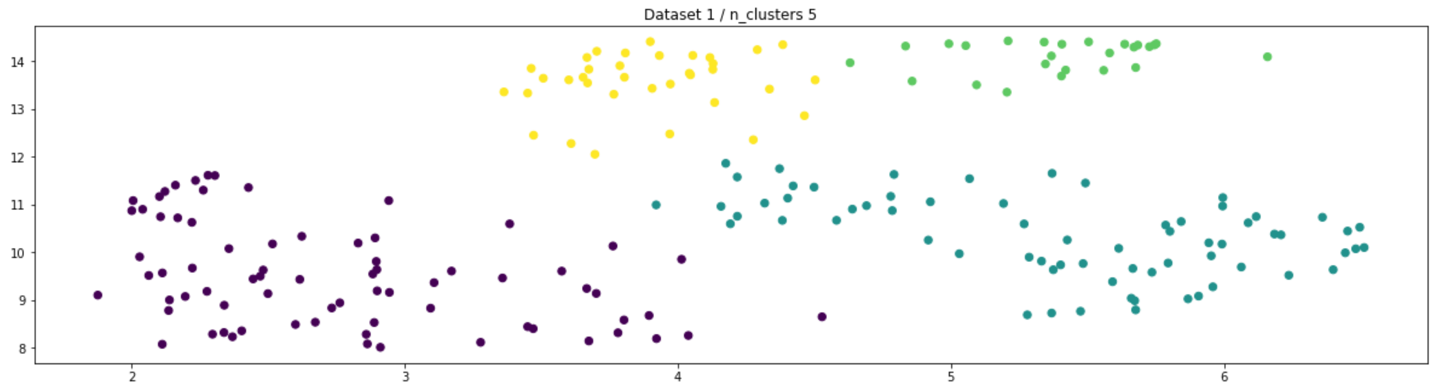

Dataset 5
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 8
Cluster 9
Posteriors
Cell index

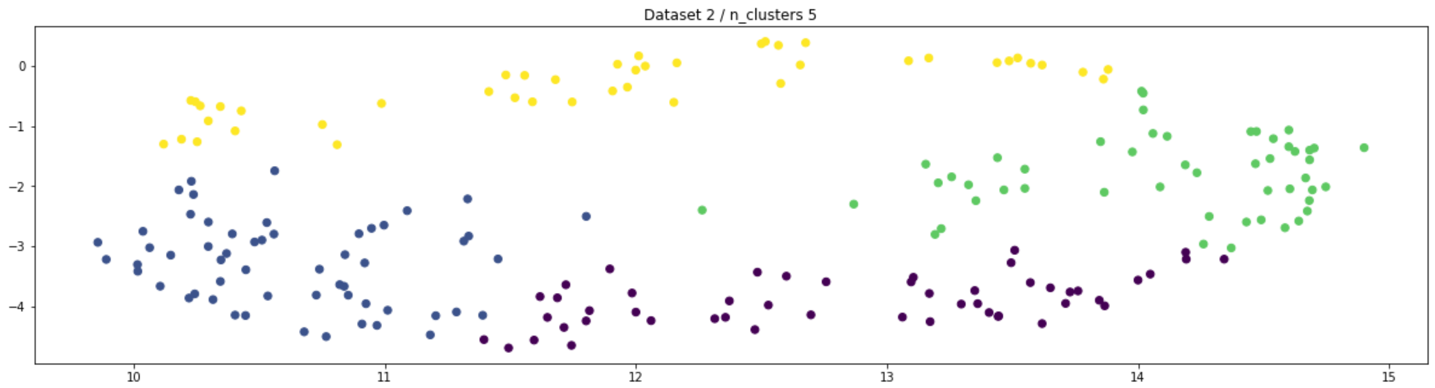
In some clusters, for example in dataset 2 with cluster 1 and 5 (plot below) we can see the transition in the probability space.

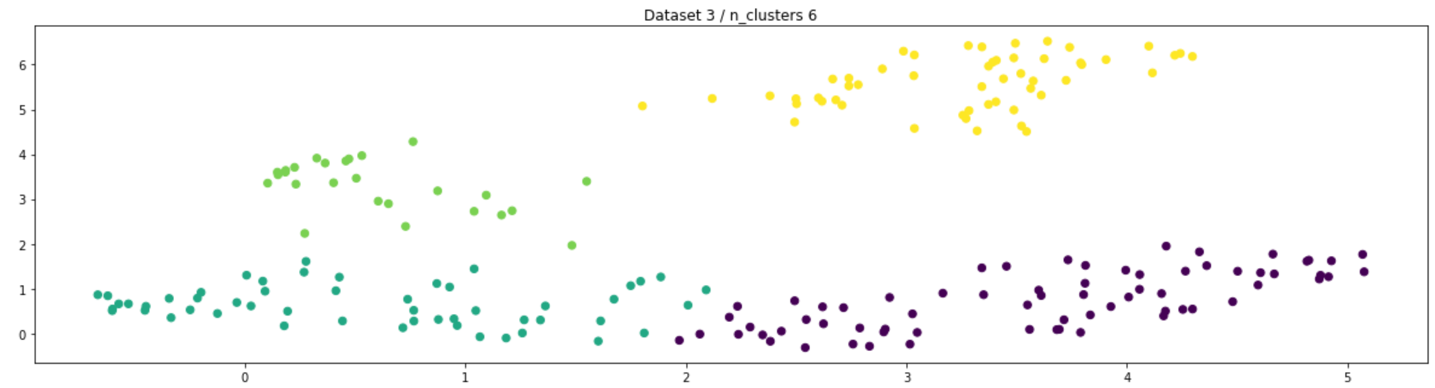


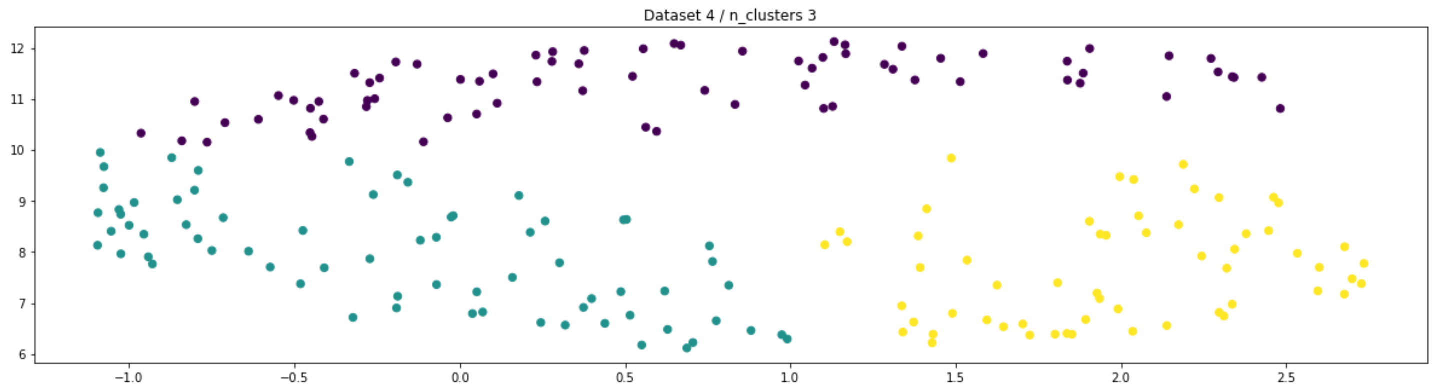
As an extra research, Bayesian Gaussian Mixture was also implemented. The difference between the Bayesian Gaussian Mixture and Gaussian Mixture Model is the number of clusters that each model separates the data. We used the same package with GMM and as parameters we defined the number of components and the number of iterations. In the table below the number of clusters the Bayesian Gaussian Mixture determined is described as also the plots of data with the relative clusters.

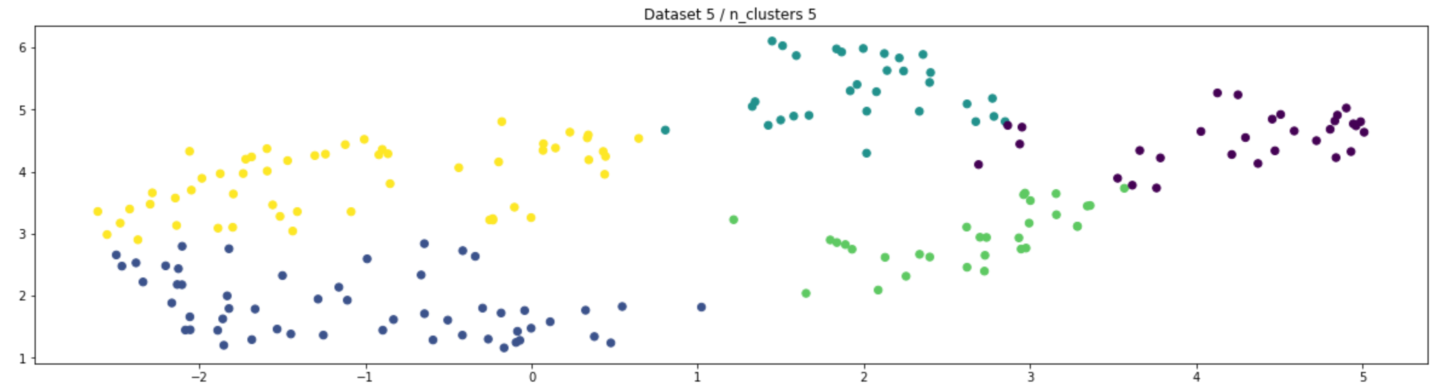
	Number of Clusters
Dataset 1	5
Dataset 2	5
Dataset 3	6
Dataset 4	3
Dataset 5	5


Dataset 1 / n_clusters 5


Dataset 2 / n_clusters 5


Dataset 3 / n_clusters 6


Dataset 4 / n_clusters 3


Dataset 5 / n_clusters 5

KMeans

Apart from the Gaussian Mixture Models, the KMeans algorithm was also used for the prediction of the optimal classes of the UMAP results. (KMeans can be seen as a special case of Gaussian mixture model with equal covariance per component.)

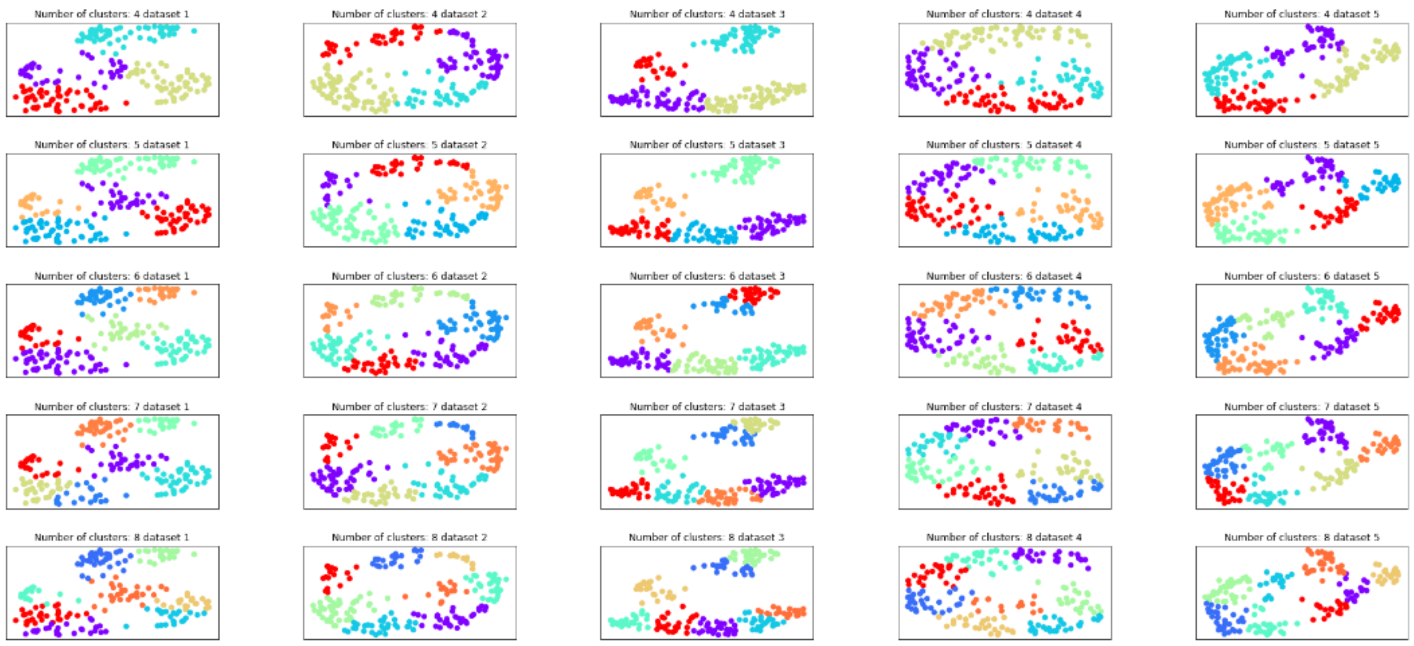
KMeans is one of the most popular Unsupervised Machine Learning Algorithms used for solving Classification Problems. KMeans segregates the unlabeled data into various groups, called clusters, based on having similar features, common patterns. It is an Iterative algorithm that divides a group of n datasets into k subgroups /clusters based on the similarity and their mean distance from the centroid of that particular subgroup/ formed.

The number of clusters that somebody can choose for the algorithm shouldn't be random. Each and Every cluster is formed by calculating and comparing the mean distances of each data points within a cluster from its centroid. Using the Within-Cluster-Sum-of-Squares (WCSS) method, the right number of clusters can be obtained. It sums the squares of distances of the data points in each and every cluster from its centroid. By plotting the several results, the sharp point of bend will be considered as the best/optimal value of K.

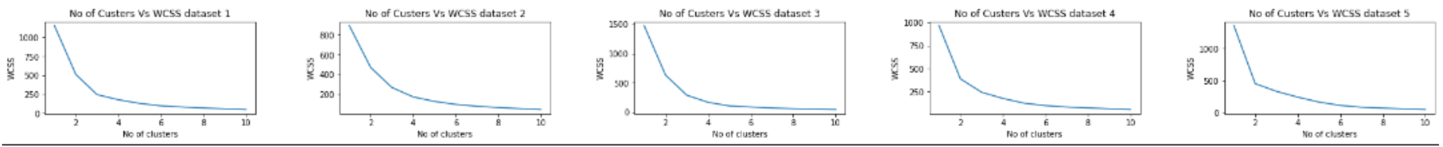
The usage of KMeans algorithm (package sklearn) is simple. The parameter that is required is the number of clusters to form as well as the number of centroids to generate.

Then, we are able to obtain the identified_clusters and the sum of squared distances of samples to their closest cluster center, weighted by the sample weights if provided.

In our case, we use for each dataset, after the UMAP dimensionality reduction, the KMeans algorithm and we found the optimal number of k for each dataset which are presented below.



Screenshot of KMeans clustering - k 4-8



Screenshot of KMeans clustering WCSS

In all datasets, the optimal number of clusters is between 4 and 5.

After performing dimensionality reduction and clustering for unsupervised learning, we can infer that we have approximately 5-6 clusters in our datasets. UMAP played a crucial role for transforming the data into lower dimensions and as a result the decision of choosing the appropriate parameters is important. After that, the Gaussian mixture model classified the results and found the optimal number of components for each data set efficiently.