Abstract:

In this assignment, we aim to develop an unsupervised learning pipeline for the analysis of single-cell gene expression data. The provided dataset consists of five synthetic datasets, each containing the expression profiles of 200 genes for 200 cells. The objective is to implement a comprehensive pipeline that encompasses dimensionality reduction, clustering, and visualization stages, utilizing various techniques such as Principal Components Analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), Uniform Manifold Approximation and Projection (UMAP), and Gaussian Mixture Modeling (GMM).

Firstly, the dimensionality reduction stage involves applying PCA, t-SNE, and UMAP to reduce the high-dimensional gene expression data to a lower-dimensional space. The selection of the optimal number of dimensions will be investigated using suitable criteria.

Next, the clustering stage employs GMM to identify the "best" number of cell states (clusters) within the reduced data. Model selection methods, specifically the Bayesian Information Criterion (BIC), will be utilized to determine the optimal GMM model with the appropriate number of components and covariance matrix structure. The outcome of this stage will be the assignment of posterior probabilities to each cell for each state, ensuring an optimal number of states.

Finally, the visualization stage focuses on presenting the results in an intuitive manner for human investigators. Various methods and techniques will be employed to visualize the inferred clusters, cell posterior distributions, and cell joint distributions. Python notebooks or R markdowns will be utilized to facilitate proper documentation and validation of the code.

The report accompanying this assignment will include detailed descriptions and discussions of each stage of the pipeline, along with the justification for the selection of packages and their specific parameters. The code will be organized into well-delineated function calls, enabling easy understanding and reproduction. Precise instructions will be provided to ensure the ease of installing dependencies and running the code, resembling a submission for a peer-reviewed journal.

Introduction:

Single-cell gene expression analysis has revolutionized the field of genomics by providing insights into the heterogeneity and cellular composition of complex biological systems. Unsupervised learning techniques play a vital role in uncovering meaningful patterns and structures within single-cell gene expression data. In this assignment, our objective is to develop a comprehensive data analysis pipeline that encompasses dimensionality reduction, clustering, and visualization stages to extract valuable information from single-cell gene expression datasets.

The provided dataset consists of five synthetic datasets, each containing the expression profiles of 200 genes for 200 cells. The challenge lies in effectively processing and analyzing this high-dimensional data to reveal underlying patterns and groupings among the cells.

Our pipeline begins with the dimensionality reduction stage, where we aim to reduce the dimensionality of the gene expression data while preserving the most relevant information. We will explore various techniques such as Principal Components Analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP). The selection of the optimal number of dimensions will be based on careful investigation and justified by suitable criteria.

Moving forward, the clustering stage utilizes Gaussian Mixture Modeling (GMM) to identify distinct cell states or clusters within the reduced-dimensional space. Model selection methods, such as the Bayesian Information Criterion (BIC), will be employed to determine the optimal GMM model with the appropriate number of components and covariance matrix structure. The assignment of posterior probabilities to each cell for each state will provide valuable insights into the clustering results.

To facilitate the interpretation and understanding of the results, the visualization stage plays a crucial role. We will leverage suitable visualization techniques to present the inferred clusters, cell posterior distributions, and cell joint distributions in an intuitive manner. The use of Python notebooks will aid in documenting and validating the code, ensuring a clear and comprehensive presentation of the results.

In conclusion, this assignment aims to develop an unsupervised learning pipeline for single-cell gene expression data analysis. By implementing dimensionality reduction, clustering, and visualization stages, we seek to extract meaningful information from the provided dataset. Through the use of appropriate techniques and careful analysis, we anticipate uncovering hidden patterns and structures within the single-cell gene expression data, contributing to a deeper understanding of cellular heterogeneity.

Method:

Our pipeline consists of three stages which are:

1\_Preprocessing the dataset:

After importing the dataset , which has 200 dimensions , which we refer to this concept known as [Curse of Dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality) that tells that a dataset with too many variables tend to become sparse makes it more difficult for machine learning models to estimate reliable outcomes. The need of observations grows exponentially as the number of variables increase.

So in order to be able to work with our dataset we need to use **Dimensionality reduction** techniques , in our pipeline we are going to use three different techniques and compare the performance of each one .

The performance and efficiency of machine learning algorithms is often hampered by the high dimensionality of

real-world datasets. Typically, the minimum number of parameters required to account for all properties of the data.

**Before starting using the dimensionality reduction methods we want to scale our data to make sure that all genes** are in the same scale. And also centerise our data so that it varies around zero. This is done by calculating the mean values of each of the variables and then subtracting these values from each measurement of a variable.(1)

In our implementation we used Z-score normalization from SKlearn (StandardScaler) which centerise it by default.

PCA (Principal Component Analysis):

PCA is a mathematical transformation that projects the data points from its current dimension to vectors of points, called components. Each component will aim to comprehend the maximum variance from the real data with the less loss of information possible.

PCA is a linear technique that works best with data that has a linear structure. It seeks to identify the underlying principal components in the data by projecting onto lower dimensions, minimizing variance, and preserving large pairwise distances.

Determining the number of Dimensions:

To determine the of dimensions I used two criterions:

* Setting the number of components that capture 95% of the variance in the dataset and you have got 109 dimensions:

pca= PCA(n\_components=0.95)

Here, we can see that the variance is well spread in many variables. There is no component with more than 4% of the variance, so it takes 109 PCs to make up to 90% of the variance from our dataset.

* Using Kaiser rule:

According to the Kaiser rule, factors or components with an eigenvalue greater than 1 should be retained. Eigenvalues represent the amount of variance explained by each factor or component. By retaining only factors or components with eigenvalues greater than 1, it is assumed that they explain a significant amount of variance in the data.

And I have got after applying kaiser rule 53 dimensions.

Visualizing the dataset using pca 2 components :

pca= PCA(n\_components=2).

t-SNE (t-distributed Stochastic Neighbor Embedding):

dimensionality reduction technique that attempts to retain the local data structure in the latent space. parametrizes the non-linear mapping between the data space and the latent space.

TSNE focus on preserve small distances by centering the gaussian distribution curve over a studied point then it measures the density of every other point in the high dimensional space under the gaussian distribution curve .

then find the similarities between points, if two points are close to each other the similarity function gives 1 , otherwise it gives 0. Student-t distribution as the heavy-tailed distribution to measure the pairwise similarities in the latent space The weights of the parametric t-SNE network are now learned in such a way that the

Kullback-Leibler divergence.

between the joint probability distributions P and Q is minimized, i.e., by minimizing. (2)

Kullback: is the distance matrix measures between the two-dimensional spaces:

A picture containing font, text, white, typography

Description automatically generated

Which a very important parametric to determine the perplexity we will use.

A diagram of normal distribution

Description automatically generated with low confidence

**Optimizing t-SNE**:

The main purpose of t-SNE is visualization of high-dimensional data. Hence, it works best when the data will be embedded on two or three dimensions.

Optimizing the KL divergence can be a little bit tricky sometimes. There are five parameters that control the optimization of t-SNE and therefore possibly the quality of the resulting embedding:

Perplexity

early exaggeration factor

earning rate

maximum number of iterations

The perplexity is defined as

where is the Shannon entropy of the conditional probability distribution. The perplexity of a -sided die is , so that

is effectively the number of nearest neighbors t-SNE considers when generating the conditional probabilities. Larger perplexities lead to more nearest neighbors and less sensitive to small structure. Conversely a lower perplexity considers a smaller number of neighbors, and thus ignores more global information in favour of the local neighborhood. As dataset sizes get larger more points will be required to get a reasonable sample of the local neighborhood, and hence larger perplexities may be required. Similarly noisier datasets will require larger perplexity values to encompass enough local neighbors to see beyond the background noise.

We used in our model the value of perplexity equal to 50.

The maximum number of iterations is usually high enough and does not need any tuning. The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor. Larger factors result in larger gaps between natural clusters in the data. If the factor is too high, the KL divergence could increase during this phase. Usually it does not have to be tuned.

A critical parameter is the learning rate. If it is too low gradient descent will get stuck in a bad local minimum. If it is too high the KL divergence will increase during optimization. A heuristic suggested in Belkina et al. (2019) is to set the learning rate to the sample size divided by the early exaggeration factor. We implement this heuristic as learning\_rate='auto' argument.

The last parameter, angle, is a tradeoff between performance and accuracy. Larger angles imply that we can approximate larger regions by a single point, leading to better speed but less accurate results

Umap:

**nonlinear** dimension-reduction algorithm that overcomes some of the limitations of t-SNE. It works similarly to t-SNE in that it tries to find a low-dimensional representation that preserves relationships between neighbors in high-dimensional space, but with an increased speed and better preservation of the data’s global structure.

UMAP is a **non-parametric** algorithm that consists of two steps: (1) compute a fuzzy topological representation of a dataset, and (2) optimize the low dimensional representation to have as close a fuzzy topological representation as possible as measured by cross entropy

There are  [main hyperparameters](https://pair-code.github.io/understanding-umap/) in UMAP that are used to control the balance between local and global structure in the final projection:

 **The number of nearest neighbors**: which controls how UMAP balances local versus global structure - low values will push UMAP to focus more on the local structure by constraining the number of neighboring points considered when analyzing the data in high dimensions. In contrast, high values will push UMAP towards representing the big-picture structure, hence losing fine detail.

 **The minimum distance between points in low-dimensional space**: which controls how tightly UMAP clumps data points together, with low values leading to more tightly packed embeddings. Larger values will make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure

the main matrix used is the distance matrix or the similarity matrix. UMAP calculates the distances or similarities between data points and constructs a high-dimensional fuzzy topological representation of the data. This representation is then optimized to find a low-dimensional embedding that preserves the global and local structure of the data. The choice of distance metric and the construction of the matrix influence the UMAP results and the quality of the embedding.

2\_ **Clustering** of the dimensionality reduced data into the “best” number of cell “states” (clusters) using Gaussian Mixture Modeling (GMM).

In machine learning, when faced with a mountain of unlabeled data, a data scientist’s first impulse is to try clustering the data. Clusters give us a way of describing data, finding commonalities between data points, and catching outliers.

But without any prior knowledge, how do we know how many clusters exist within the data?

Most clustering techniques require that we choose a fixed number of clusters. An algorithm like k-means will then find the centers of these k different clusters.

The Bayesian Information Criterion (BIC) or BIC criterion:

The BIC balances the number of model parameters k and number of data points n against the maximum likelihood function, L. We seek to find the number of model parameters k that minimizes the BIC

The BIC agrees with our initial visual estimation. It also tells us that a larger number of clusters would also fit the data fairly well, but at the cost of having to introduce more parameters.

Expectation-Maximization (EM) algorithm. The EM uses an iterative method to calculate and recalculate the parameters of each cluster (distribution), i.e., mean, variance/covariance, and size

At the outset, the model initializes a specified number of clusters with a set of parameters that can either be random or specified by the user. Smart initialization options are also available in some implementations (e.g., sklearn’s implementation of GMM by default uses kmeans to initialize clusters).

Initialization the clusters is a very important step in implementation of GMM, because GMM’s final result tends to be quite sensitive to the initial starting parameters.

Expectation (E-step) — for each data point, a “responsibility” r is calculated, which is, in simple terms, a probability of that data point belonging to a cluster c. This is done for each point with regard to each cluster.

Maximization (M-step) — then “responsibilities” are used to recalculate the mean, covariance, and size of each cluster (distribution).

3. **Visualization** of your results (clusters inferred, cell posteriors, cell joint distributions)

The Bonus part :

The second proposed pipeline is to do the dimensionality reduction used autoencoder and then use Density-Based Clustering (DBSCAN) algorithm which doesn’t need to initialize it with number of clusters.

An autoencoder is an artificial neural network model that is trained to reconstruct its input data. It is primarily used for unsupervised learning and dimensionality reduction tasks. The autoencoder architecture consists of an encoder and a decoder component.

autoencoders (Hinton and Salakhutdinov, 2006) can learn the non-linear mappings that are required for such embeddings, but they primarily focus on maximizing the variance of the data in the latent space.(2)

We used the encoder part of the autoencoder to extract the latent space representation of your input data. This lower-dimensional representation can be further used for various purposes, which is here clustering.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based clustering algorithm widely used for discovering clusters in spatial datasets. Unlike traditional clustering algorithms, DBSCAN does not assume that clusters have a specific shape or size and is capable of finding clusters of arbitrary shapes.

DBSCAN offers several advantages, including the ability to discover clusters of varying shapes and sizes, robustness to noise and outliers, and not requiring a predefined number of clusters. However, it does have some limitations, such as the sensitivity to the epsilon and min\_samples parameters and difficulty in handling datasets with varying densities.

To use DBSCAN in Python, you can utilize the scikit-learn library, which provides an implementation of the DBSCAN algorithm.

BSCAN object with specified values for the epsilon (eps) and minimum samples (min\_samples) parameters. We fit the DBSCAN model to the data and obtain the cluster labels. Finally, we visualize the clusters using a scatter plot.

epsilon and min\_samples is crucial and may vary depending on the dataset and desired clustering results. It's often necessary to experiment with different parameter values to find the appropriate settings for DBSCAN.

Discussion and results:

Libraries and Packages:

* Sklearn
* Pandas
* Numpy
* Matplot
* Seaborn
* Matplotlib
* UMAP
* TSNE
* PCA
* Optuna
* Seaborn
* plotly

FUNCTIONS:

importdata(t='dataset1.csv'):

this function uses pd.read\_csv function to read the dataset and saved in Dataframe using pandas dataframe.

preprocessing(df):

this function scale the dataset using StandardScaler().

def visualizedata(scaled\_data, df\_original):

visualizing the first two components of pca using (px.scatter) and it uses the cells as a labels .

apply\_pca(dataset, n\_components):

function applies pca in the dataset and it returns pca reduced data and a plot of pca components also the number of components.

The inputs of this function:

The dataset we need to reduce its dimensionality

And the number of components we need to capture .

hotmap(pca\_data)

check the co-relation between our scaled dataset using a heat map. For this, we have already imported the seaborn. The correlation between various features is given by the corr() function and then the heat map is plotted by the heatmap() function.

kasierrule(pca):

function applies the kaiser rule to determine the number of dimensions and return the number of dimensions and the final pca with the plots to visualize it .

kasier\_visualization\_plot(centered\_dataset):

function visualize how the kaiser rule work .

tsne\_perplexity\_curve(centered\_dataset)

function plots a curve labeled by Divergence and Perplexity Values to reveal the relationship between the diverfence value and perplexity.

applying\_tse\_for\_dif\_matrices(centered\_dataset,perplexity)

function to visualize the effect of choosing the type of matrix distance in the representative of datasets.

umap\_visual\_matrices(umap\_results):

applying\_UMAP\_for\_dif\_matrices(centered\_dataset,n\_neighbors,min\_dist):

function used to visualize the effect of using different types of matrices in presenting the dataset in the laten space.

dim\_red\_UMAP\_(centered\_dataset,n\_neighbors,min\_dist,metric,n\_components)

after visualization and optimatization applying the result of the optimization step in the umap function to obtain the reduced dimensional space using umap.

pplying\_tnse\_(centered\_dataset,n\_components,perplexity,metric)

after visualization and optimatization applying the result of the optimization step in the umap function to obtain the reduced dimensional space using tsne.

objective(trial) :

function for tuning the GMM clusiteifier to find its best parameter depending on the Bic score.

Bic (range\_of\_clusters,data, covariance\_type, init\_params):

It’s for visualization purpose to see find best optimal number of clusters since it’s a very crucial decision

.

A screenshot of a computer program

Description automatically generated with low confidence

Using this piece of code to make better decisions in what is the optimal number of clusters,

Depending on another criterion whish is the Silhouette score,

Silhouette score checks how much the clusters are compact and well separated.

#he more the score is near to one, the better the clustering is.

The higher Silhouette score is the best number of clusters we have.

I recommend that after finding the best parameters for the GMM and after seeing the Bic score curve we can then take the number of clusters the number which score the lowest score in the Bic curve at the same time it scores the higher score in the Silhouette score.

GMM\_(X,n\_components,init\_params,n\_init,max\_iter,covariance\_type)

This function applies the GMM classifier with the best parameters which we found previously.

visual\_GMM\_clusters(labels):

to visualize the clusters in 2 D space

posterior\_(model, clust ,labels):

Compute posterior probabilities and then show it using heatmap.

def plot\_scatters(data, gmm, title='', method='Dim', name='')

visualizing the posterior in plot.

Bonus part:

autoencoder\_(data):

using autoencoder to find the reduced data

bscan\_(encoded\_data)

For dataset 1:

Results for dataset 1:

A picture containing screenshot, text, plot, line

Description automatically generated

A picture containing text, screenshot, plot, line

Description automatically generated

A graph of heatmap

Description automatically generated with low confidence

The colour scale on the side of the heatmap helps determine the magnitude of the co-relation. In our example, we can clearly see that a darker shade represents less co-relation while a lighter shade represents more co-relation. The diagonal of the heatmap represents the co-relation of a feature with itself – which is always 1.0, thus, the diagonal of the heatmap is of the highest shade

A picture containing text, screenshot, rectangle, font

Description automatically generated

A picture containing screenshot, line, plot, slope

Description automatically generated

A graph with a red line

Description automatically generated with low confidence

A picture containing text, screenshot

Description automatically generated

A picture containing text, screenshot

Description automatically generated

From the previous plot I found that using 'correlation matrix with Perplexity Values equal to 50 is the best.

Perplexity Values (n\_neighbors): This parameter determines the number of nearest neighbors used by the t-SNE algorithm to construct the neighborhood graph. It controls the local neighborhood structure and influences the resulting embedding. A higher value may capture more global structure, but at the cost of increased computational complexity. In this case, it is set to 50, indicating that the t-SNE algorithm will consider the 50 nearest neighbors when constructing the neighborhood graph.

Here we couldn’t optimize using automatic way because we want to visualize the data taking into account many other values as explained in the methods section.

We notice that using 'correlation matrix make the dataset concentrated and not spark and also . It captures the linear relationship between the variables and can be useful since we working with gene expression data.

A graph with red dots

Description automatically generated with low confidence

A picture containing text, screenshot

Description automatically generated

After visualization we choose to initialize the umap function with those parameters:

1. centered\_dataset: This parameter represents the input dataset that has been preprocessed and centered around zero. It is the dataset on which the UMAP algorithm will be applied.
2. n\_components: This parameter specifies the number of dimensions in the reduced space. In this case, it is set to 5, indicating that the UMAP algorithm will project the dataset into a five-dimensional space.
3. n\_neighbors: This parameter determines the number of nearest neighbors used by the UMAP algorithm to construct the neighborhood graph. It controls the local neighborhood structure and influences the resulting embedding. A higher value may capture more global structure, but at the cost of increased computational complexity. In this case, it is set to 0.2, which represents a fraction of the total number of samples. The actual number of neighbors will be calculated based on this fraction.
4. metric: This parameter defines the distance metric used to measure the similarity between data points in the UMAP algorithm. In this case, the 'correlation' metric is used, which computes the correlation coefficient between the feature vectors of the data points. It captures the linear relationship between the variables and can be useful when working with gene expression data.
5. random\_state: This parameter controls the random number generator used by the UMAP algorithm for reproducibility. In this case, it is set to 42, ensuring that the same random state is used each time the UMAP algorithm is applied, leading to consistent results.

Clustering :

We used the reduced dimensionality data umap which have been known as a best algorithm in this domain.

Finding the optimal number of cluster and after running the trail function the best parameters the function found is:

Best Parameters: {'n\_components': 4, 'covariance\_type': 'full', 'init\_params': 'kmeans'}

Best BIC Score: -20527.59493307513

Which mean 4 clusters ,

Let’s visualize the Bic curve:

A graph with a red line

Description automatically generated with low confidence

A picture containing line, plot, diagram, screenshot

Description automatically generated

I found that 4 clusters is the best value taking into account the two criterions

A picture containing screenshot, text, diagram, plot

Description automatically generated

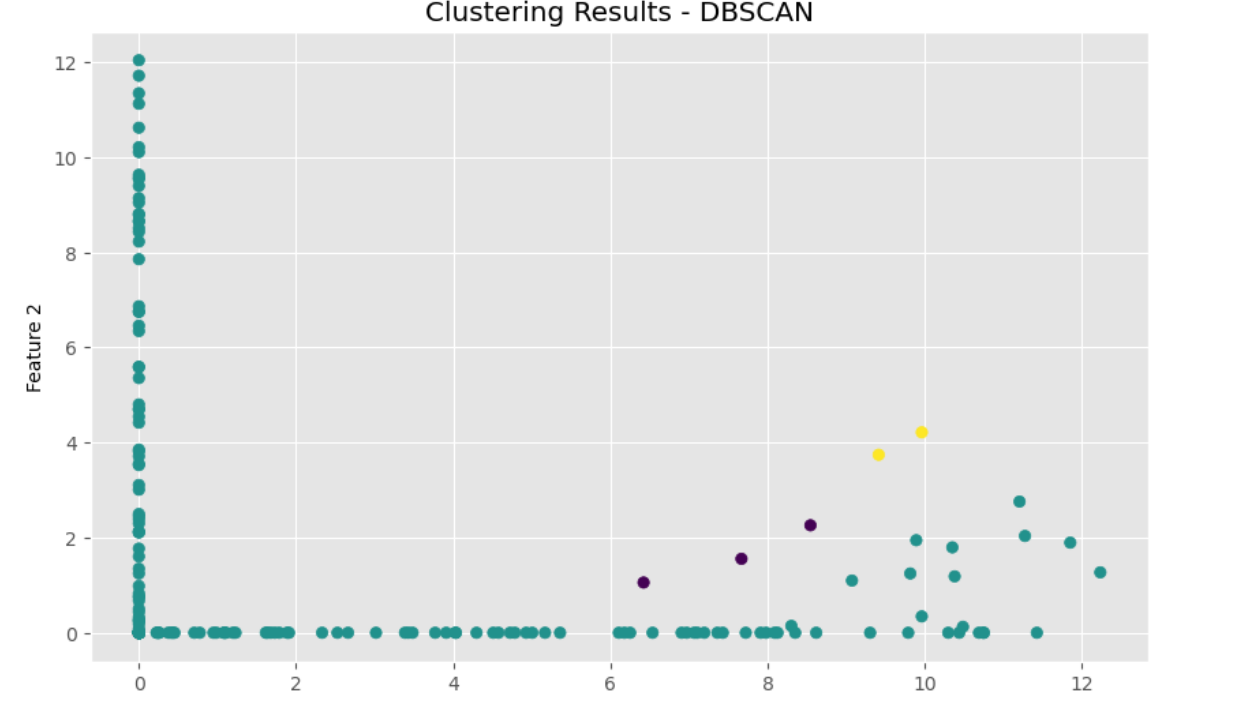
A picture containing text, screenshot, parallel, number

Description automatically generated

A picture containing text, screenshot, plot, diagram

Description automatically generated

The clusters using the new pipeline:



For the second dataset2:

A picture containing line, plot, diagram, text

Description automatically generated

A picture containing text, screenshot, rectangle, design

Description automatically generated

A graph of heatmap

Description automatically generated with low confidence

Original number of features: 200

Reduced number of features: 109

A picture containing line, plot, screenshot, diagram

Description automatically generated

Since the first data and the same are much similar to each other

We will choose different distances matrix

Here I am going to use 'mahalanobis’ matrix and compare the result

Best Parameters: {'n\_components': 2, 'covariance\_type': 'tied', 'init\_params': 'kmeans'}

Best BIC Score: -15469.913381670693

A graph with a red line

Description automatically generated with low confidence

A picture containing plot, line, diagram, screenshot

Description automatically generated

From the plots it says the 2 cluster is the significant number of clusters:

A chart with yellow and purple dots

Description automatically generated with low confidence

From the plot we can say that 3 clusters is a candidate also ,

Let’s try 3 clusters:

A picture containing screenshot, text, colorfulness, diagram

Description automatically generated

A picture containing text, screenshot, line, diagram

Description automatically generated

​

cell\_joint\_distributions

[array([0.89022482, 0. , 0. ]),

array([0. , 0.9871114, 0. ]),

array([0. , 0. , 0.99917578])]

A picture containing text, screenshot, plot, diagram

Description automatically generated

For the new pipeline :

A picture containing screenshot, text, plot, diagram

Description automatically generated

And the all datasets very much similar to each other and we imply the same analysis.

Conclusion :

In conclusion, this assignment focused on developing an unsupervised learning pipeline for the analysis of single-cell gene expression data. The pipeline consisted of three main stages: dimensionality reduction, clustering, and visualization.

In the dimensionality reduction stage, Principal Components Analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP) were employed. These techniques allowed us to reduce the high-dimensional gene expression data into lower-dimensional representations, capturing the most relevant information. The selection of the optimal number of dimensions was carefully investigated, considering criteria such as explained variance and neighborhood preservation.

The clustering stage utilized Gaussian Mixture Modeling (GMM) to identify distinct cell states or clusters within the reduced-dimensional space. The optimal GMM model, in terms of the number of components and covariance matrix structure, was determined using the Bayesian Information Criterion (BIC). This enabled the assignment of posterior probabilities to each cell, providing insights into the clustering results and aiding in the identification of cell states.

The visualization stage played a crucial role in presenting the results in an intuitive manner. Various techniques were employed to visualize the inferred clusters, cell posterior distributions, and cell joint distributions. Python notebooks or R markdowns were used to document and validate the code, facilitating clear and comprehensive presentation of the results.

In conclusion, this assignment showcased the power of unsupervised learning techniques in uncovering meaningful insights from single-cell gene expression data, paving the way for future discoveries and advancements in the field of genomics. But from this pipeline and from using dimensionality reduction we can gain insights about genes and cells relation and build a hypothesis about it but we can’t prove our hypothesis, it need to tests with more advanced and reliable way.

From searching and reading we understood how proposed reduction dimensonality method work and how they perform and which one is much better for nonlinear dataset.

In my new pipeline I used the encoder but umap and tsne outperform more in this field because autoencoder fails in retaining the local structure of the data.