ECSE556 Homework 1 Report

Romain Bazin
McGill University
romain.bazin@mail.mcgill.ca

September 29, 2023

1 Introduction

The objective of this assignment is to explore *transcriptomic data*, which encompasses the expression levels of numerous genes across hundreds of cell lines, through the application of various Machine Learning techniques.

It's important to highlight that several preprocessing steps were implemented on the dataset before any computations to prevent data leakage or biases. Initially, we fixed the random seed to ensure reproducibility. Next, we transformed the dataset by transposing it, resulting in gene expressions being organized in columns and samples in rows. Subsequently, we divided the dataset into two portions: the first part, constituting 80% of the total data, was designated for visualization, clustering, and model training, while the remaining 20% was set aside for evaluation purposes. Finally, the training subset was utilized to train a standard scaler, which was then applied directly to the training data to standardize the features, resulting in all gene expressions having a mean of 0 and a standard deviation of 1.

All the code necessary to reproduce the results of this report is available here.

2 Dimensionality Reduction

2.1 Visualization

Using standard libraries like scikit-learn and umap-learn, we plot the features, reduced into a 2D space, with three different dimensionality reduction methods : PCA, UMAP, t-SNE.

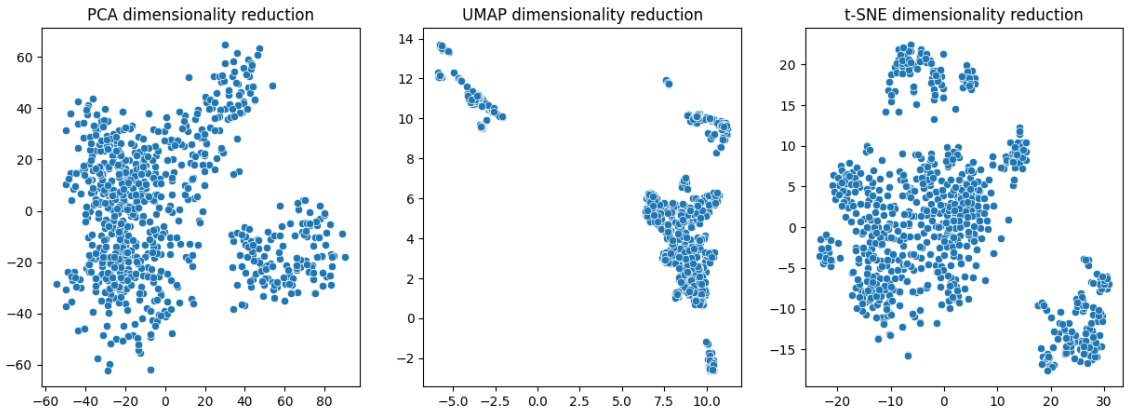

Figure 1: 2D dimensionality reduction of gene expression data. Three different methods were used, from left to right : PCA, UMAP, t-SNE.

On figure 1, we observe that the three methods yield very different 2D representations. This was quite expected as the methods largely differ in nature. Yet, on each figure, we can roughly identify three clusters (represented on figure 2). We note that it's easier to identify the clusters with the UMAP and t-SNE methods. UMAP yields an even denser 2D representation that t-SNE.

All figures are pretty different, but it could be argued that t-SNE are quite similar in aspect, compared to UMAP.

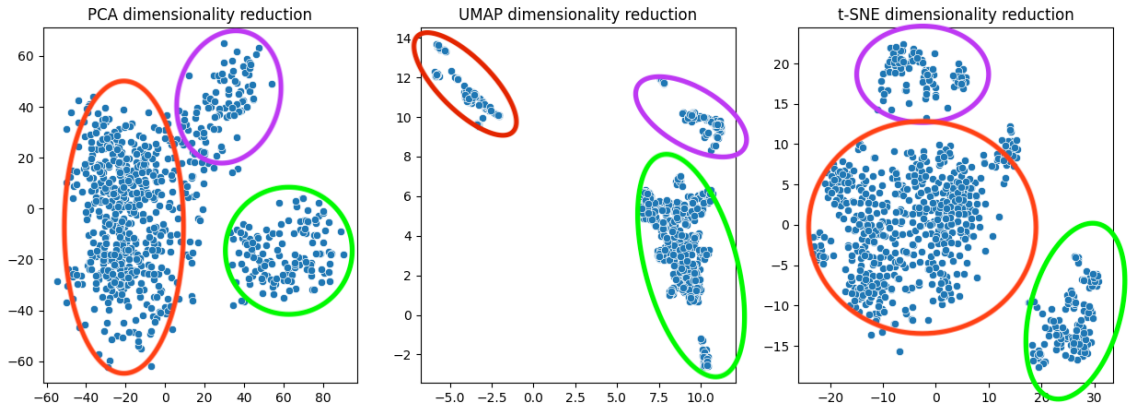

Figure 2: Clusters roughly identified from different 2D projection methods.

2.2 Prons and cons

PCA is a linear dimensionality reduction method that focuses on maximizing variance. It finds orthogonal linear combinations of features called principal components to reduce dimensionality. Its advantages include efficiency for high-dimensional data and the preservation of global structure and linear relationships. However, it assumes linear relationships, which may not capture complex non-linear patterns.

t-SNE is a non-linear dimensionality reduction technique that minimizes divergence between probability distributions representing pairwise similarities in high and low dimensions. It excels in preserving local structure and visualizing complex, non-linear patterns. Nonetheless, t-SNE is computationally expensive, sensitive to random initialization, and lacks direct feature interpretability.

UMAP is another non-linear dimensionality reduction method that constructs a low-dimensional representation by modeling the underlying manifold structure of data. It successfully preserves both local and global structures, making it a balanced choice. UMAP is more computationally efficient than t-SNE but requires careful parameter tuning and offers less interpretability compared to PCA.

To summarize, PCA is suitable for linear data with an emphasis on preserving global structure, t-SNE is ideal for visualizing complex, non-linear relationships but is computationally intensive, and UMAP strikes a balance between preserving local and global structures while being more computationally efficient than t-SNE, though it still requires parameter tuning. The choice depends on the specific characteristics and goals of your data analysis.

3 Clustering

3.1 Agglomerative clustering vs K-means clustering

Using the Agglomerative Clustering (AC) method and K-Means (KM) method (scikit-learn implementations, default parameters), we identify 3 different clusters (n_clusters = 3) from the data.

To compare the similarity between each of these clusters, we compute the *Jaccard similarity* scores (size of the intersection over size of the union of two sets) on each pair of clusters. A Jaccard score close to 1.0 means there's a huge overlap between the two sets, hence they are similar, contrary to a score of 0 that entails there's no overlap between the clusters, which means they are totally disctinct from each other. The results can be found in table 1.

The Jaccard scores show that the clusters with the same index are mostly very similar. Only Cluster-2, from the Agglomerative Clustering method has a similarity split between Cluster-0 and Cluster-2 from the K-means method, still its similarity with Cluster-2 remains high (83.9%). We also note that some clusters have absolutely no overlap between the two methods, like Cluster-0 (AC) with Cluster-1 (KM); Cluster-1 (AC) with Cluster-2 (KM); Cluster-2 (AC) with Cluster-1 (KM).

	Cluster-0	Cluster-1	Cluster-2
Cluster-0	0.951128	0	0.004769
Cluster-1	0.004545	0.977612	0
Cluster-2	0.030817	0	0.839161

Table 1: Jaccard similarity scores between all clusters. Agglomerative Clustering clusters are on the rows and K-Means clusters are on the columns.

Overall, the way each method creates the clusters is very similar, as shown by the *Rand Score* and *Adjusted Rand Score*, respectively equal to 0.947 and 0.893. Also the clusters are not named the same, we see on figure 3 that the two algorithms have a pretty similar clustering distribution.

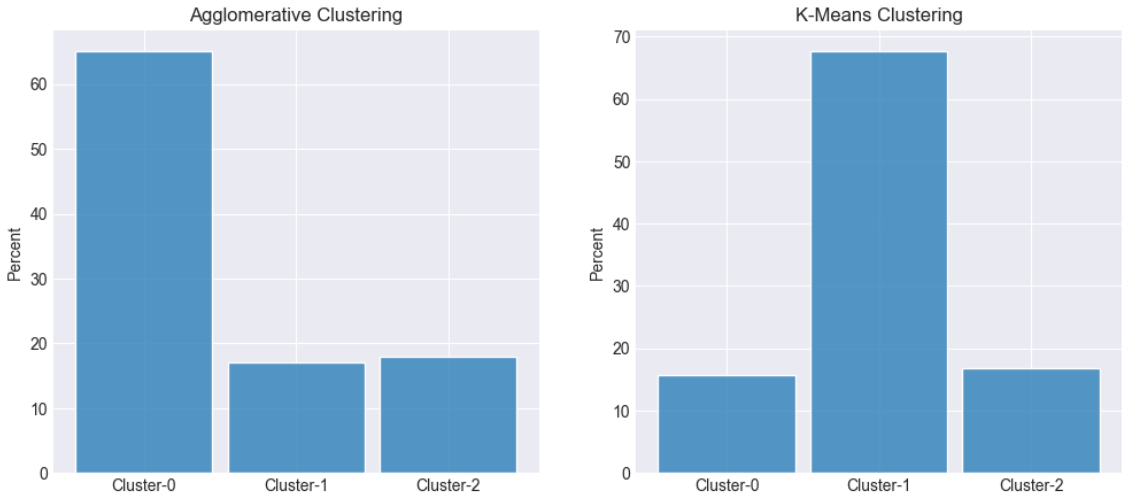

Figure 3: Respective clustering distributions of the Agglomerative and K-Keans clustering methods.

3.2 Variations of agglomerative clustering

To evaluate the impact of different parameters in the Agglomerative Clustering method, we conducted an experiment with a key variation from our previous approach. This time, we altered the "linkage" parameter, switching it from "ward" to "average," and employed either the *cosine* or *euclidean* distance metric.

To compare the resulting clusters, we followed the same methodology as before by calculating the *Jaccard similarity* between clusters generated using the two variations. The results are summarized in table 2.

	Cluster-0	Cluster-1	Cluster-2
Cluster-0	0.358515	0	0
Cluster-1	0.319285	0.003968	0.003968
Cluster-2	0.321383	0	0

Table 2: Jaccard similarity scores between all clusters. The two methods used are variations of the Agglomerative Clustering method. The cosine distance variation is on the rows and the euclidean variation is on the columns.

From the table, it is evident that the clusters generated using the cosine variation exhibit a notably similar score, around 33%, with Cluster-0 from the euclidean variation. However, they exhibit minimal similarity with the other clusters. Furthermore, the *Rand Score* and *Adjusted Rand Score* between the two variations are approximately 0.3352 and 0.0001 (rounded to four decimal places), respectively. The first score indicates that only one-third of the total number of corresponding cluster pairs match, which are primarily pairs involving Cluster-0 from both methods. The second metric, with its very low score, offers more profound insights, as it suggests

a fundamental dissimilarity between the two clustering methods when considering their inherent distributions. This can be more clearly observed on figure 4.

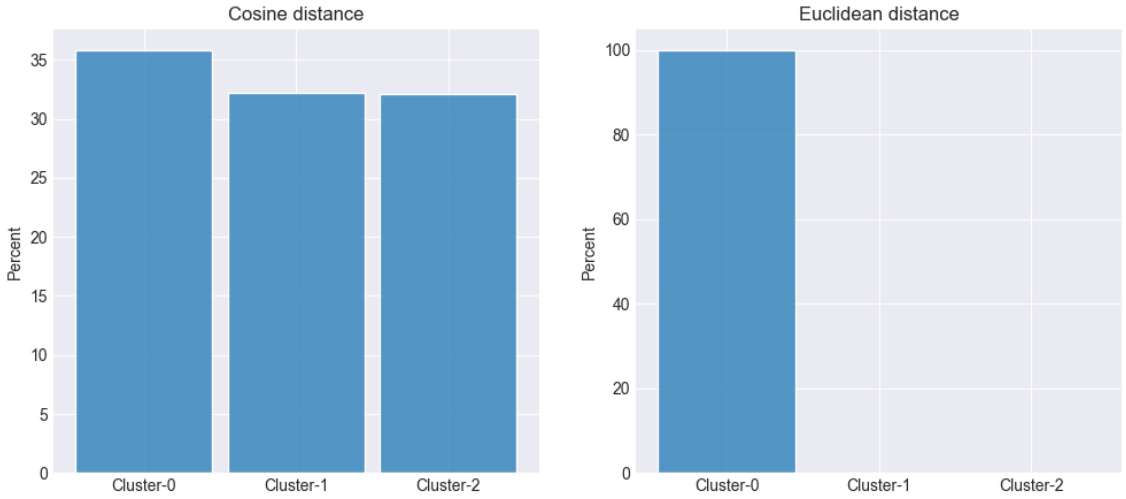

Figure 4: Respective distribution of the two aggloremative clustering variations.

The figure illustrates that the euclidean variation results in the discovery of only one cluster. Consequently, since the clusters generated by the cosine variation are relatively evenly balanced, they all share an approximate 33% similarity with Cluster-0 from the second method. In this context, the Rand and Adjusted Rand scores become more transparent. With three clusters, if one were to randomly assign a sample to a cluster with a 33% probability, the Rand Score would reflect a 33% correctness rate. In contrast, the Adjusted Rand Score adjusts for this initial random assignment, eliminating the influence of random assignment to clusters with equal probabilities.

The variance in clustering outcomes between cosine and Euclidean distances when applied to a high-dimensional dataset with 13,000 features can be attributed to the sensitivity of these metrics to high-dimensional spaces and variations in feature scales. Cosine distance, emphasizing vector direction, yields well-balanced clusters, whereas Euclidean distance, considering both magnitude and direction, leads to a dominant cluster with empty others. This divergence arises from the "curse of dimensionality" in Euclidean space, where data points become increasingly scattered, posing challenges for clustering. Employing feature scaling and dimensionality reduction techniques may mitigate Euclidean distance's sensitivity, but the choice of distance metric should align with the data's characteristics and the clustering objectives.

In summary, within this specific scenario, the Euclidean variations result in an unsatisfactory representation where all samples are grouped into a single cluster. While it is not advisable to manipulate clustering algorithms to force a desired outcome, it is expected that a clustering algorithm should ideally produce multiple clusters. The coherence and meaningfulness of these clusters should be evaluated subsequently.

4 Regression

In this section, we employed LASSO regression (implemented using scikit-learn) as our regression method. We varied the L_1 regularization parameter, denoted as α, while keeping the maximum number of iterations fixed at 10,000. For the first experiment, we used the same scaled features as in the previous section, which were divided into training and test sets. For the second experiment, the process described was done using the raw dataset. In both cases, our target label was the drug response for the "Doxorubicin" drug, and it was also split into training and test sets for the first experiment.

The first experiment involved training the LASSO model with a fixed training set and different values of the regularization parameter α selected from the set [0.01, 0.1, 0.3, 0.5, 0.9]. LASSO is employed to yield sparse linear solutions, where some coefficients are set to zero during regression. Consequently, we can interpret a coefficient equal to zero as a feature not being considered during inference. For each model trained, we tracked the count of selected features for each α value. The results of this experiment are presented in Figure 5. The figure illustrates that increasing the value of the regularization coefficient leads to a reduction in the number of selected features.

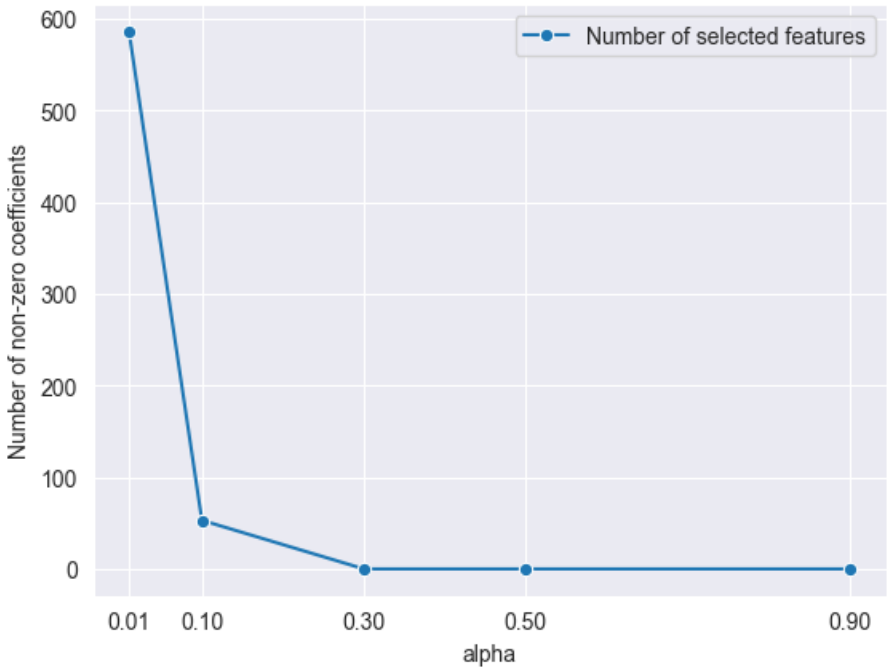

Figure 5: Number of selected features by each model trained with a different value of α. A feature being selected means the corresponding coefficient in the linear regression was non-zero.

In the second experiment, we aimed to determine which parameter yielded the best model, indirectly identifying the most important features. To ensure there was no data leakage or selection bias affecting the experimental results, we employed a nested cross-validation process. This process not only alters the portions of the dataset used for training and validation but also designates a separate portion as the test set. To achieve this, we utilized scikit-learn's KFold, GridSearchCV, and cross_validate functionalities. Initially, we created an inner split with 3 folds and an outer split with 4 folds using KFold. Subsequently, we constructed a GridSearchCV instance with an estimator that included a StandardScaler and a Lasso regression estimator (utilizing the Pipeline class). The parameter grid for this search included only the various values of α we intended to explore, and the optimization fold was defined as the inner fold established earlier. Finally, we trained the GridSearchCV instance using cross-validation over the outer fold, which enabled us to compute Mean Squared Error (MSE) and Spearman Rank values on the test set.

The results consistently indicated that the optimal value for α across all inner splits was 0.1. On the test sets of all outer splits, the average MSE was 2.568, and the average Spearman Rank was 0.297. Additionally, for each outer fold split, we calculated the average training MSE and Spearman Rank values, which were 1.476 and 0.741, respectively. These values were notably higher than those on the test folds, suggesting potential overfitting of the model. Therefore, it would be advisable to explore the optimization of other hyperparameters such as the maximum number of iterations or the tolerance parameter.