Machine Learning in Network Biology
Homework 1

Dr. Amin Emad

Gian Favero: 261157396

October 6th, 2023

Contents

1 Introduction 3

2 Part 1: Dimensionality Reduction 3
2.1 Theory . . . 3
2.2 Implementation . . . 4
2.3 Discussion of Results . . . 8

3 Part 2: Clustering 9
3.1 Theory . . . 9
3.2 Implementation . . . 9
3.3 Analysis of Agglomerative Clustering . . . 11

4 Part 3: Regression 13
4.1 Regularization Analysis . . . 13
4.2 Nested Cross-Validation . . . 15
4.3 Implementation . . . 15
4.4 Cross-Validation Results . . . 15
4.5 Comparison to Baseline . . . 16
4.6 Discussion . . . 16

A Clustering (Non-Normalized Data) 18

B Regression (Non-Normalized Data) 19

1 Introduction

Transcriptomic data refers to a collection of RNA molecules, or transcripts, present in cell tissues at a reference time. Typically, these molecules are generated in a cell through the process of transcription of DNA into RNA. This data contains insights into the expression levels of genes in a cell which can be used to predict various items like disease, stimuli response, and development, among others [1].

Several classical machine learning approaches can be used when analyzing transcriptomic data. These include dimensionality reduction, clustering, and regression. All three of these particular approaches will be explored in this assignment using popular Python libraries to gain some insight into the data provided. Gene expression data was provided alongside this assignment(`gdsc_expr_postCB.csv`) and drug response data (`gdsc_dr.csv`).

2 Part 1: Dimensionality Reduction

Dimensionality reduction is a technique used to reduce the number of features in a dataset. This is done by projecting the data onto a lower dimensional space and finding a new, simpler set of features that can be used to represent the original data. Memory usage, computational efficiency, and visualization are all advantages of dimensionality reduction.

Several dimensionality reduction techniques are explored in this assignment. These include Principal Component Analysis (PCA), UMAP, and t-SNE algorithms.

2.1 Theory

Principal Component Analysis

The following is adapted from Goodfellow et al. [2].

For a set of m data points $\mathbf{x}^{(i)} \in \mathbb{R}^n$, PCA finds a lower dimensional representation of the data by representing it as a code vector $\mathbf{c}^{(i)} \in \mathbb{R}^l$ where $l < n$. The goal of PCA is to find a function f such that $\mathbf{c}^{(i)} = f(\mathbf{x}^{(i)})$ and a decoding function g which provides $\mathbf{x} \approx g(f(\mathbf{x}))$. This is simply done using a decoding matrix, $\mathbf{D} \in \mathbb{R}^{n \times l}$ which has columns that are orthogonal and have a unit norm. Following this, the code vector is given by:

$$\mathbf{c} = f(\mathbf{x}) = \mathbf{D}^T \mathbf{x}$$

and the reconstructed input is given by:

$$\mathbf{x} \approx g(\mathbf{c}) = \mathbf{D}\mathbf{D}^T \mathbf{x}$$

Using a closed-form or numerical optimization method, the decoding matrix can be found by minimizing the reconstruction error, $||\mathbf{x} - \mathbf{D}\mathbf{D}^T\mathbf{x}||_2^2$ across each point in the dataset.

l is typically the only parameter selected by the user. For visual purposes, l is typically chosen as 2 such that the compressed data can be plotted on a Cartesian plane.

t-SNE

t-SNE was proposed by Van der Maaten et al. [3] as a stochastic technique for visualizing high-dimensional data. At a high-level, the algorithm maps each data point to a location on a 2D or 3D map that can reveal the structure at many scales. The resulting map effectively reveals low-dimensional manifolds on which the data points lie - the more similar the data points are, the closer they are placed to each other on the map.

t-SNE algorithms are incorporated into the `scikit-learn` Python library. The algorithm is capable of optimizing a dimensionality reduction on a dataset based on a number of parameters selected by a user. Some of these include the number of dimensions and the perplexity, which dictate the manner in which the algorithm structures the lower-dimension manifolds in which the data lies.

UMAP

UMAP is a novel non-linear manifold learning technique by McInnes et al. [4]. UMAP is a scalable algorithm that is comparable to t-SNE in visualization, but superior in preservation of global structure and runtime.

UMAP algorithms have been abstracted into a Python library, `umap-learn`, which is capable of optimizing a dimensionality reduction on a dataset based on a number of parameters selected by a user. Some of these include the number of dimensions, the number of neighbors and the minimum distance between points, which play roles in the generalization of the algorithm, the trends identified (local or global), and the distinction between clusters.

2.2 Implementation

Brief data pre-processing was done before implementing the different dimensionality reduction techniques. Commonly used libraries such as `numpy` and `matplotlib` were imported, before the data was loaded into a numpy array, stripped of its first row (instance headers) and its first column (feature headers) and transposed such that the data was in the form of a $n \times m$ matrix where n is the number of instances and m is the number of features.

The dimensionality reduction techniques were then implemented. The `scikit-learn` library was used to implement PCA and t-SNE, while the `umap-learn` library was used to implement UMAP. Specifically, the `sklearn.decomposition.PCA`, `umap`, and `sklearn.manifold` classes were used for PCA, UMAP, and t-SNE respectively. Please refer to the linked documentation for more information on the class parameters and their default values.

First, the techniques were implemented with their default parameters with the exception of the number of dimensions, which was set to 2 for all techniques. The results of this implementation are shown in Figure 1.

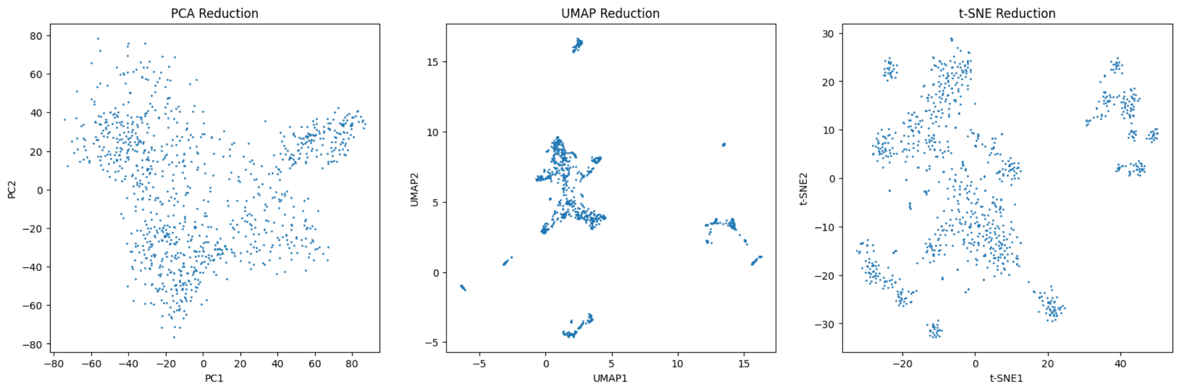

Figure 1: PCA (left), UMAP (middle), and t-SNE (right) reduction using default parameters.

A quick implementation with the default parameters of each technique gives some insight into the nature of the performance. As seen in Figure 1, distinct regions of point clusters have been generated in both the UMAP and t-SNE plots. The PCA plot, however, does not show any distinct clusters and instead shows a large cloud of points with no discernible structure. Knowing that both UMAP and t-SNE are non-linear techniques, it can be deduced that perhaps linear methods, such as PCA, are not the most suitable dimensionality reduction approaches for this dataset.

In addition to the visual inspection of the plots, the runtime of each technique was also recorded when run on a machine with an Intel i7-8750H processor and 16 GB of RAM. The runtime of each technique is shown in Table 1.

Method	**Runtime (s)**
PCA	0.383
UMAP	5.850
t-SNE	3.916

Table 1: Runtime of each technique using default parameters.

A point to note is that while both UMAP and t-SNE have generated distinct clusters, the clusters are not identical in shape or density. UMAP has produced denser clusters that are closer to the origin, while t-SNE has produced a similar amount of clusters that are more spread out, both in terms of point-to-point and cluster-to-cluster. To examine the non-linear techniques further, parameters of interest were changed in isolation to analyze their effect on the resulting plots.

Effect of Scaling

The effect of scaling the data was analyzed by implementing the dimensionality reduction techniques on the scaled data. The data was scaled using the `sklearn.preprocessing.StandardScaler` function. The results of this implementation are shown in Figure 2.

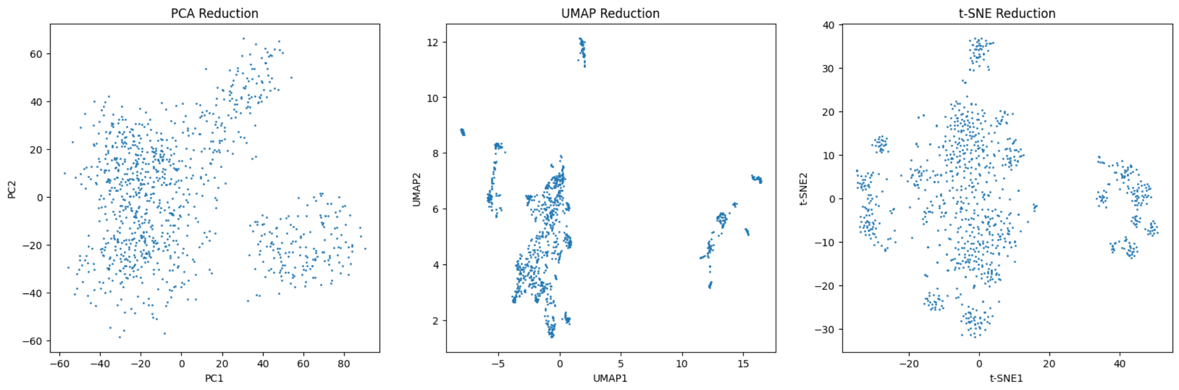

Figure 2: PCA (left), UMAP (middle), and t-SNE (right) reduction using scaled feature data.

Scaling the data had a significant effect on the PCA plot but less of an effect on UMAP and t-SNE. The PCA plot now shows two denser regions of points, but none to the extent of the clusters of data points grouped by the other two algorithms. The UMAP and t-SNE plots are visually similar to the plots generated using the unscaled data.

This micro-experiment demonstrates the importance of scaling the data in certain scenarios, like when conducting a PCA. In doing so, it can be ensured that the data is not biased towards features with larger variances.

UMAP Parameter Analysis

A sequence of reductions with varying number of neighbors in UMAP is observed in Figure 3.

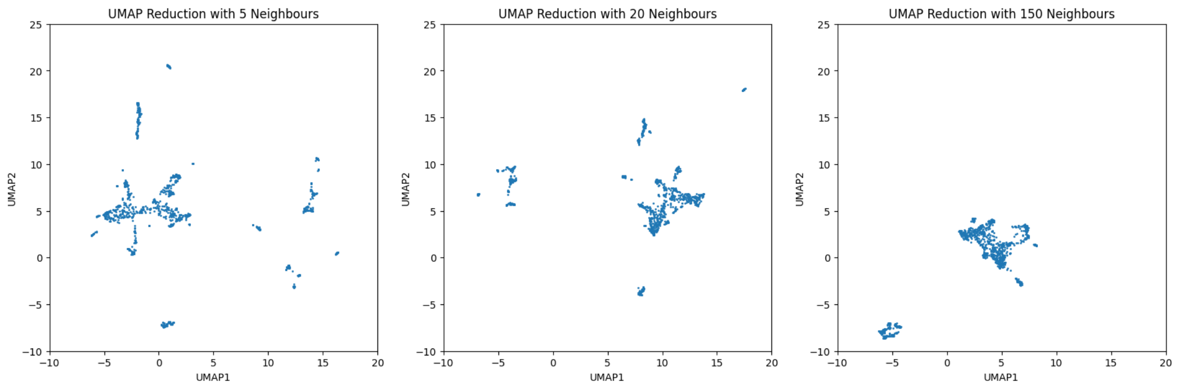

Figure 3: UMAP reduction of the data varying the number of neighbors.

Visually, adjusting the number of neighbors considered in a UMAP reduction provides a different insight into the relationships between data points. As seen in Figure 3, the number of clusters identified by the algorithm decreases as the number of neighbors increases. It can be discerned then, that considering more neighbors captures broader trends in the data, while considering fewer neighbors captures more local trends.

A sequence of reductions with varying minimum distance in UMAP is observed in Figure 4.

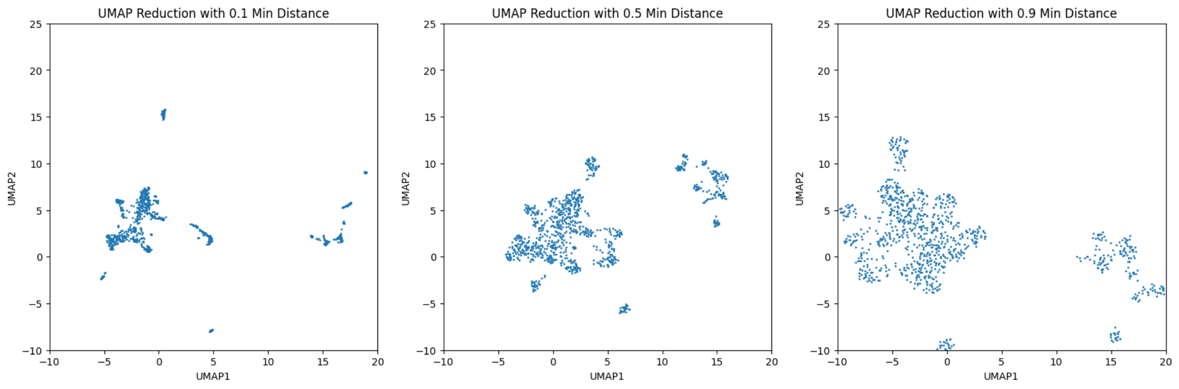

Figure 4: UMAP reduction of the data varying the minimum distance.

From inspection, the minimum distance parameter seems to have a spread effect on the clusters and data points. As the minimum distance increased, the clusters reduced in number and became more spread out. This is likely due to the fact that the minimum distance parameter is used to control the effective minimum distance between points. As the minimum distance increases, the algorithm is forced to spread out the clusters to meet the minimum distance requirement.

Depending on the objective of the dimensionality reduction, both the number of neighbors and the minimum distance parameters can be used to control the generalization of the algorithm. If the goal is to identify broad trends in the data, then a higher number of neighbors and a lower minimum distance should be used. If the goal is to identify local trends in the data, then a lower number of neighbors and a higher minimum distance should be used.

t-SNE Parameter Analysis

A sequence of reductions with varying perplexity in t-SNE is observed in Figure 5.

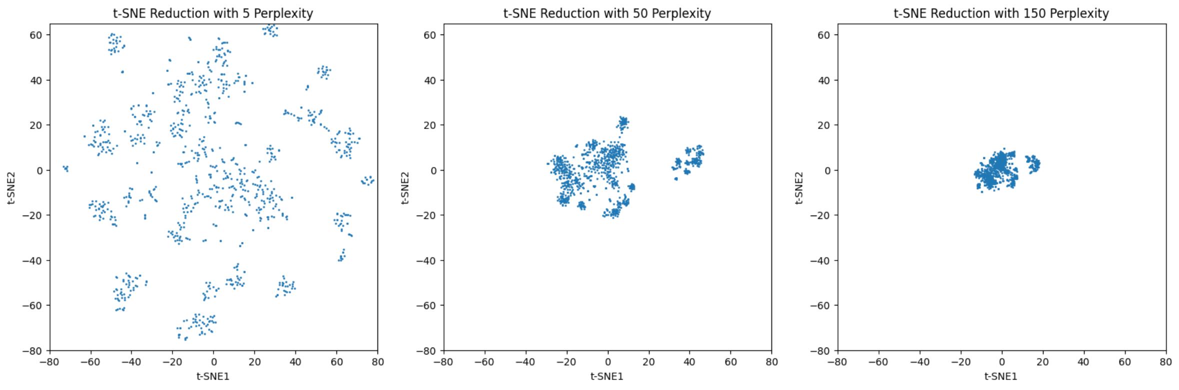

Figure 5: t-SNE reduction of the data varying the perplexity.

Similar to the number of neighbors parameter in UMAP, the perplexity parameter in t-SNE controls the number of neighbors considered when reducing the dimensionality of the data. As seen in Figure 5, the number of clusters identified by the algorithm decreases as the perplexity increases. It can be similarly discerned then that considering more neighbors captures broader trends in the data, while considering fewer neighbors captures more local trends.

Again, depending on the objective of the dimensionality reduction, the perplexity parameter can be used to control the generalization of the algorithm. If the goal is to identify broad trends in the data, then a higher perplexity should be used. If the goal is to identify local trends in the data, then a lower perplexity should be used.

2.3 Discussion of Results

1. **Do you see different clusters forming (visually)? How much correspondence exist between clusters that you find in different methods?**

   Different clusters formed visually in the UMAP and t-SNE plots. The clusters in the UMAP plot were denser and closer to the origin, while the clusters in the t-SNE plot were more spread out. The clusters in the PCA plot were not visually discernible. On the scaled data, it is hard to define exactly how many clusters were formed in the PCA, UMAP, and t-SNE plots, but it appears that there were approximately 3 clusters in each.

   Varying the parameters of both the UMAP and t-SNE algorithms resulted in different clusters being formed. The number of clusters identified by the algorithms decreased as the number of neighbors or perplexity increased. This is due to the fact that considering more neighbors or a higher perplexity captures broader trends in the data, while considering fewer neighbors or a lower perplexity captures more local trends. Therefore, the non-linear techniques are surely capable of producing high correspondence, but the parameters must be tuned to achieve this correspondence. However, neither UMAP nor t-SNE had striking correspondence with the PCA plot.

   It should be mentioned also that UMAP and t-SNE have some element of randomness involved (UMAP contains a random initialization, t-SNE contains a random seed). Therefore, the results of the algorithms will never be fully reproducible, or have high correspondence with itself when run multiple times.

2. **What is the difference between the three methods above? Use pros/cons to compare them against each other.**

   PCA is a fast algorithm (was the fastest of the three), is computationally efficient, scaled well with this large dataset, and is highly interpretable. However, PCA assumes that linear relationships are present in the data, and is therefore not suitable for datasets with non-linear relationships. In these cases, a loss of local structure can occur. It appears that the linear assumption did not hold particularly well with this dataset, as the PCA plot did not show any discernible preserved structure. Finally, PCA was sensitive to scaling of the data, while the other two algorithms were not. When the data was scaled, PCA was performed more effectively, showing its sensitivity to bias towards features with larger variances.

   UMAP is a non-linear, highly customizable, dimensionality reduction technique that is capable of reducing the dimensionality of a dataset while preserving the local and global structure of the data. In this implementation, UMAP was experimentally found to be the slowest of the three algorithms as a result of its high degree of complexity (however, in theory, should scale better than t-SNE with more data). This complexity also leads to lesser interpretability as well. However, it was well suited for this dataset as it was able to capture the non-linear relationships in the data and produce dense clusters from the data capturing both local and global trends. Finally, UMAP was not sensitive to scaling of the data, but is based on a random initialization and therefore is not fully reproducible.

t-SNE is another non-linear dimensionality reduction technique that was effective in identifying relationships in the gene expression data. Like UMAP, it appeared to be effective in preserving local structures in the data, but was less effective in preserving global structures. Further, t-SNE was faster than UMAP in this experiment, but could not generate clusters as dense on a 2D plot. However, like UMAP, t-SNE is tunable and can be used to capture local trends in the data effectively. Finally, the stochastic nature of t-SNE makes it difficult to reproduce results.

3 Part 2: Clustering

3.1 Theory

Agglomerative Clustering

Agglomerative clustering is a clustering approach that works by first partitioning a dataset into singleton clusters, and then iteratively merging the clusters together until there is only one cluster remaining [5]. There are several parameters that govern the algorithm: linkage, distance metric, and, obviously, the number of clusters. Linkage refers to the method used to calculate the distance between clusters, while the distance metric refers to the method used to calculate the distance between points.

K-Means Clustering

K-means clustering is a traditional unsupervised algorithm used to partition a dataset into k clusters. The algorithm works by first initializing k centroids, iteratively assigning each point to the cluster with the closest centroid, and then updating the centroids based on the new cluster assignments until a steady state is reached. The algorithm is sensitive to the initial centroid positions, and therefore, the algorithm is typically run multiple times with different initialization to ensure that the global minimum is reached.

3.2 Implementation

The `sk.learn.cluster.AgglomerativeClustering` function was used to implement agglomerative clustering. Initially, the function was used with its default parameters with the exception of the number of clusters, which was set to 3. With its default settings, the agglomerative clustering algorithm uses a "ward" linkage, which minimized the variance of clusters being merged, and Euclidean affinity.

The `sklearn.cluster.KMeans` function was used to implement K-means clustering with a specification of 3 clusters. Given that the initial clusters are assigned randomly, the following results are based on an individual run of the algorithm and are therefore are not fully reproducible without setting a random seed.

As seen in the previous section, scaling the data is imperative for distance-based algorithms which are heavily influenced by variance. Therefore, the data was normalized before implementing the clustering algorithms.

Jaccard Similarity with Default Parameters

The computed clusters of each algorithm were compared using the Jaccard similarity metric implemented via the `sklearn.metrics.jaccard_score` function. An iterative approach was conducted to compute the similarity of the nine permutations of the three algorithms. The results of this approach are shown in Table 2. Note that the rows represent the clusters computed by the agglomerative algorithm, while the columns represent the clusters computed by the K-means clustering algorithm.

K-means / Agg.	**Cluster 1**	**Cluster 2**	**Cluster 3**
Cluster 1	0.952	0.032	0.001
Cluster 2	0.006	0.821	0.000
Cluster 3	0.000	0.006	0.982

Table 2: Jaccard similarity of the clusters using default parameters.

The Jaccard similarity metric is a measure of the similarity between two sets. The metric is defined as the size of the intersection of the sets divided by the size of the union of the sets. In this case, the sets are the clusters identified by the algorithms. The Jaccard similarity metric is a value between 0 and 1, where 0 indicates no similarity and 1 indicates perfect similarity. From Table 2, it can be seen that the clusters identified by the algorithms are not identical, however there is a high degree of similarity. Note that the values were rounded to 3 decimal places for readability.

Rand Index with Default Parameters

The computed clusters of each algorithm were compared using the Rand index metric implemented via the `sklearn.metrics.adjusted_rand_score` function. Adjusting for 3 decimal places, the Rand-index of the two algorithms at default parameters was approximately **0.946**, on a scale from 0 to 1, with 1 indicating perfect correspondence.

Adjusted Rand Index with Default Parameters

The computed clusters of each algorithm were compared using the adjusted Rand index metric implemented via the `sklearn.metrics.adjusted_rand_score` function. Adjusting for 3 decimal places, the adjusted Rand-index of the two algorithms at default parameters was approximately **0.893**, on a scale from -1 to 1, with -1 indicating disagreement, 0 indicating randomness, and 1 indicating a perfect correspondence.

Clearly, both indexes suggest that the clusters identified by the algorithms are not identical, however there is a high degree of similarity.

Visualization

The clustering algorithms were also compared visually.

Cluster	Agglomerative	K-Means
Cluster 1	633	655
Cluster 2	179	156
Cluster 3	167	168

Table 3: Number of Samples in Each Cluster (Default)

The algorithms have approximately the same number of points in each cluster, as seen in Table 3

Visually, the clusters formed were analyzed by plotting the data points in each cluster on a 2D plot reduced via PCA. The results of this analysis are shown in Figure 6.

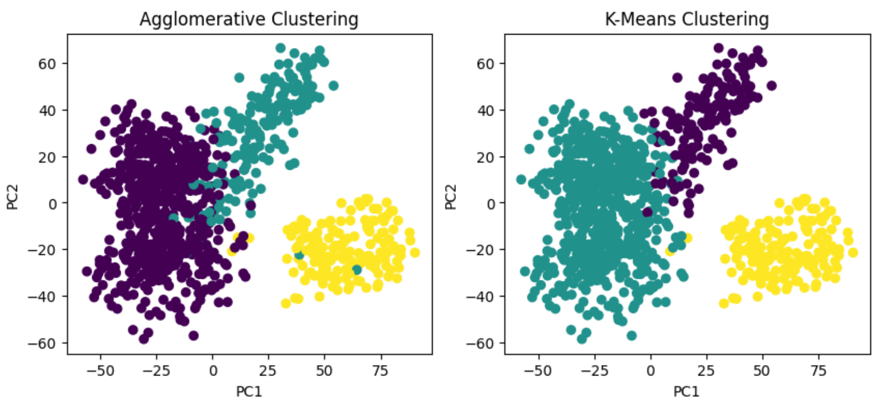

Figure 6: PCA reduction of the data with clusters identified by agglomerative clustering (left) and K-means clustering (right).

The color mapping was chosen to highlight the clusters identified by the algorithms. Confirmed visually, the clusters formed by the two algorithms are nearly identical.

3.3 Analysis of Agglomerative Clustering

Agglomerative clustering was implemented separately with average linkage and Euclidean and cosine distance metrics. An additional analysis done on non-normalized data can be found in Appendix A.

Jaccard Similarity with Average Linkage

The Jaccard similarity of the clusters identified by the algorithms with average linkage is shown in Table 4.

Cosine \ Euclidean	**Cluster 1**	**Cluster 2**	**Cluster 3**
Cluster 1	0.351	0.000	0.000
Cluster 2	0.353	0.000	0.003
Cluster 3	0.296	0.003	0.000

Table 4: Jaccard similarity of the clusters using average linkage.

It can be seen that the clusters formed by the average linkage algorithms are not very pairwise similar to each other according to the Jaccard score. Notably, cluster 1 from the algorithm using the Euclidean distance metric is nearly one-third similar to each of the clusters identified by the algorithm using the cosine distance metric.

Rand Index with Average Linkage

Adjusting for 3 decimal places, the Rand-index of the two algorithms with average linkage was approximately **0.336**, on a scale from 0 to 1, with 1 indicating perfect correspondence.

Adjusted Rand Index with Average Linkage

Adjusting for 3 decimal places, the adjusted Rand-index of the two algorithms with average linkage was approximately **0.0**, on a scale from -1 to 1, with -1 indicating disagreement, 0 indicating randomness, and 1 indicating a perfect correspondence.

Visualization

Table 5 shows the distribution of samples in each cluster for the average linkage algorithms.

Cluster	Euclidean	Cosine
Cluster 1	977	343
Cluster 2	1	346
Cluster 3	1	290

Table 5: Number of Samples in Each Cluster (Average Linkage)

The clusters formed by the algorithms with average linkage were analyzed visually by plotting the data points in each cluster on a 2D plot reduced via PCA. The results of this analysis are shown in Figure 7.

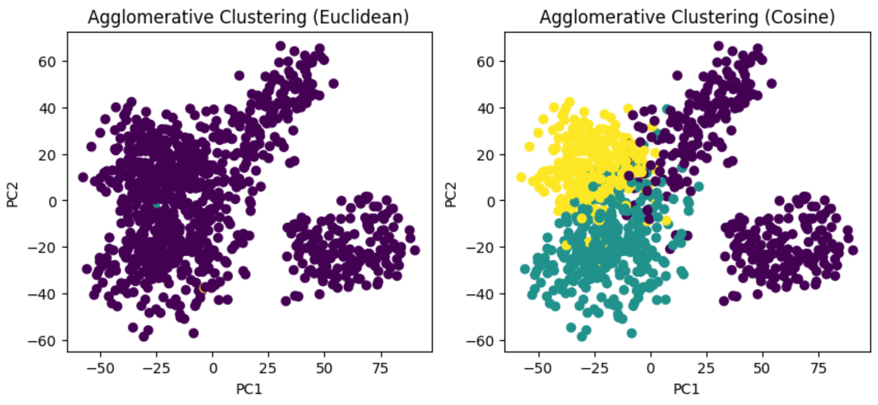

Figure 7: PCA reduction of the data with clusters identified by agglomerative clustering with average linkage.

The agglomerative clustering algorithm using the Euclidean distance metric placed nearly every point in the same cluster! Agglomerative clustering with average linkage and the cosine distance metric seems to have done a fairly good job at forming three uniform clusters in the data. Each cluster contains roughly one-third of the data points and visually appear to be somewhat related to each other in the PCA plot. The algorithm using the Euclidean distance metric, however, has formed one massive cluster containing nearly all of the data points, with the remaining two clusters containing only one point each.

We now have an explanation for the ambiguity in the Jaccard scores, Rand-index, and adjusted Rand-index. Correspondingly, this massive cluster is approximately one-third similar to the three normal-sized clusters identified using the cosine distance metric, which is aptly represented in the Jaccard score and Rand-index metrics.

This scenario shows the relative strengths and weaknesses of the two distance metrics with high-dimensional data. Given that the feature data observed is of a very high-dimension, the "Curse of Dimensionality" is very clearly at play here, which causes the Euclidean distance metric to converge to a constant value. Hence, forming distinct clusters becomes a real challenge. The cosine distance metric, however, measures the cosine of the angle between two vectors. This reflects the similarity in orientation or direction of the vectors in a high-dimensional space and makes it insensitive to magnitude and sparse data. In the case of gene expression data, which is sparse and high-dimensional, the cosine distance metric is clearly a much better choice than the Euclidean distance metric.

4 Part 3: Regression

4.1 Regularization Analysis

In this section, target data was introduced to the dataset in the form of drug response data. The drug response vector corresponding to "Doxorubicin" was aligned with the gene expression data, with all instances being aligned with the same drug response value, and instances with no drug response data being removed. The feature data was then normalized using for convergence of the regression model. A non-normalized version of the feature data was also used, and the results are shown in Appendix B.

A LASSO regression model, which is a linear model with L1 regularization, was then fit to the data using the `sklearn.linear_model.Lasso` function. The model was fit using a range of alpha values in [0.01, 0.1, 0.3, 0.5, 0.9], where the optimal weight vector is given by:

$$w^* = \min_w \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha ||w||_1$$

An analysis was done to examine how the value of alpha affects the number of features selected by the LASSO regression model. This was implemented by iterating over the alpha values and recording the number of non-zero coefficients in the weight vector. The results of this analysis are shown in Figure 8.

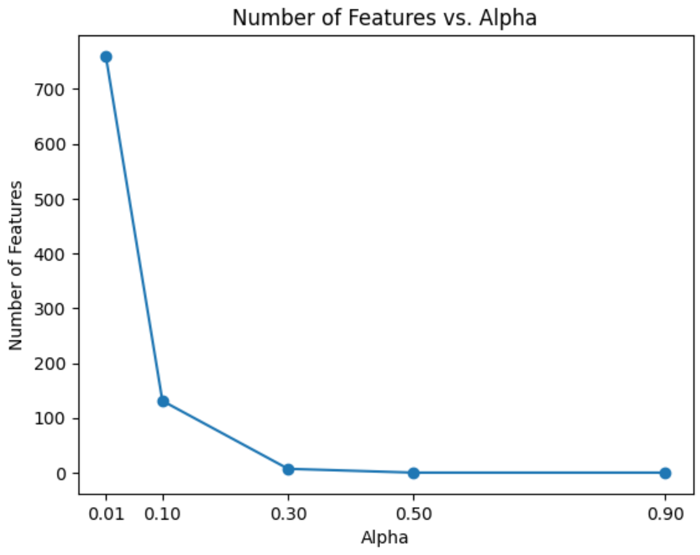

Figure 8: Number of non-zero coefficients in the weight vector as a function of alpha.

The results are also shown in Table 6.

Alpha	Number of Non-Zero Coefficients
0.01	760
0.1	131
0.3	7
0.5	0
0.9	0

Table 6: Number of non-zero coefficients in the weight vector as a function of alpha.

As alpha increases, the sparsity of the solution increases dramatically. This is conventional with theory, as more weight is given to the L1 norm penalty and thus the optimal solution will be more sparse. With this dataset, it is seen that an alpha value of 0.5 results in a weight vector with no non-zero coefficients.

4.2 Nested Cross-Validation

Nested cross-validation is a technique used for robustly evaluating the performance of machine learning models while simultaneously optimizing the hyperparameters associated with the models. Initially, inner and outer loop cross-validation (CV) folds are established employing stratification to ensure balanced representation. In this case, it was specified that the outer loop was to have 4 folds, while the inner loop was to have 3. Subsequently, a parameter grid is defined to be iterated over during hyperparameter optimization, which occurs in the inner loop of the CV process. In this case, the hyperparameter alpha was tested at values [0.01, 0.1, 0.3, 0.5, 0.9].

In the outer loop, the data is partitioned into 4 distinct training (75% of the data) and test sets (25% of the data). Within each outer fold, the training data is partitioned again into 3 folds of training (66%) and validation sets (33%). The inner loop is used to compare the performance of each model at the specified hyperparameter values. In this case, it was known that the LASSO estimator was to be used, so correspondingly model performance was evaluated in the inner loop using MSE.

The alpha value that results in the lowest MSE in the inner loop cross validation is then used in the model to be evaluated in the outer loop. The model is re-fit on the entire outer fold training set, and evaluated on the unseen test set using MSE and Spearman Rank Correlation. This process is repeated for each outer fold, resulting in 4 MSE and Spearman Rank Correlation values. The average of these values is then taken as the final performance metric for the model.

4.3 Implementation

The procedure outlined above was implemented in Python using Scikit-learn. The `sklearn.model_selection.KFold` method was used to define the outer and inner loops desired. First, the feature data was split via the outer loop object and then a for-loop was constructed to iterate over each outer fold. Within each outer fold, the data was partitioned into train and validation subsets to be used in the inner loops.

The `sklearn.model_selection.GridSearchCV` object parameterized by a LASSO estimator, the parameter grid, and the inner loop fold object was then used to perform the inner loop CV process. The grid search object automatically optimized the hyperparameters of the estimator based on the inner loop CV, refit the model on the entire outer fold training set, and then was able to return various metrics such as the optimal hyperparameter value and the best estimator model. The best estimator was then used to predict the test set of the outer fold with its performance evaluated by MSE and Spearman Rank Correlation.

This process was repeated for each outer fold, and the average MSE and Spearman Rank Correlation were computed. Note that the Spearman Rank Correlation was computed using the `scipy.stats.spearmanr` function.

4.4 Cross-Validation Results

The results of the nested cross-validation process at each outer loop fold are shown in Table 7.

Outer Fold	Alpha	Mean Squared Error	Spearman Correlation	p-value
1	0.1	2.308	0.323	1.1×10^{-6}
2	0.1	2.258	0.341	2.0×10^{-7}
3	0.1	2.930	0.317	1.8×10^{-6}
4	0.1	2.306	0.344	2.0×10^{-7}

Table 7: Results of the nested cross-validation process.

The Spearman Rank Correlation at each outer loop fold was found to be statistically significant at a significance level of less than 0.05. Thus, the average MSE and Spearman Rank Correlation of a LASSO regression model were found to be **2.451** and **0.331** respectively.

Each inner loop CV observed that the optimal alpha value was 0.1. Given the consistency, it seems that the optimal alpha value is not sensitive to the folds in the training data.

4.5 Comparison to Baseline

To further evaluate the LASSO regression model, its performance in MSE was compared to some basic baseline models - a mean estimator and a median estimator. The mean estimator simply predicts the mean of the training data for all test instances, while the median estimator simply predicts the median of the training data for all test instances. The dummy models were evaluated using the same outer loop CV process as the LASSO regression model, and the average MSE was computed. Note that no Spearman Rank Correlation was computed for the dummy models as they predict a constant value for all test instances.

The results of this comparison are shown in Table 8.

Model	MSE	Spearman Rank Correlation
Mean	2.780	NaN
Median	2.798	NaN
LASSO	2.451	0.331

Table 8: Comparison of models.

The LASSO regression model was found to outperform both the mean and median dummy models in MSE and Spearman Rank Correlation.

4.6 Discussion

Based on the MSE and Spearman Rank Correlation results alone, it can be difficult to tell the true effectiveness of the LASSO regression model. While the LASSO regression model outperformed the dummy models, the MSE and Spearman Rank Correlation values do not seem particularly good in an absolute sense. For example, the Spearman Rank Correlation value of 0.331 indicates that there is a positive correlation between the predicted and actual drug response values, but the correlation is not particularly strong.

A more detailed analysis of state-of-the-art models would be required to determine the true effectiveness of the designed LASSO regression model.

References

[1] "Transcriptome Fact Sheet." [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/Transcriptome-Fact-Sheet

[2] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[3] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[4] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction." [Online]. Available: http://arxiv.org/abs/1802.03426

[5] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," Sep. 2011, arXiv:1109.2378 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1109.2378

A Clustering (Non-Normalized Data)

A similar analysis was conducted on the non-normalized data to examine the effect of scaling on the clusters produced by the agglomerative clustering algorithm. The results of this analysis are shown below.

Cosine \ Euclidean	**Cluster 1**	**Cluster 2**	**Cluster 3**
Cluster 1	1.000	0.000	0.000
Cluster 2	0.000	1.000	0.000
Cluster 3	0.000	0.000	1.000

Table 9: Jaccard similarity of the clusters using average linkage and Euclidean and cosine distance.

Interestingly, when the data is not normalized, the clusters produced by the average linkage algorithm are identical regardless of the distance metric used. Correspondingly, the Rand-Index and adjusted Rand-Index were found to be **1.0**. Therefore, it can be concluded that the clusters produced by the average linkage algorithm are identical regardless of the distance metric used for this non-normalized dataset.

The visualization of the clusters produced by the average linkage algorithms are shown in the Figure and Table below.

Cluster	Euclidean	Cosine
Cluster 1	972	972
Cluster 2	1	1
Cluster 3	6	6

Table 10: Number of Samples in Each Cluster (Non-Normalized Average Linkage)

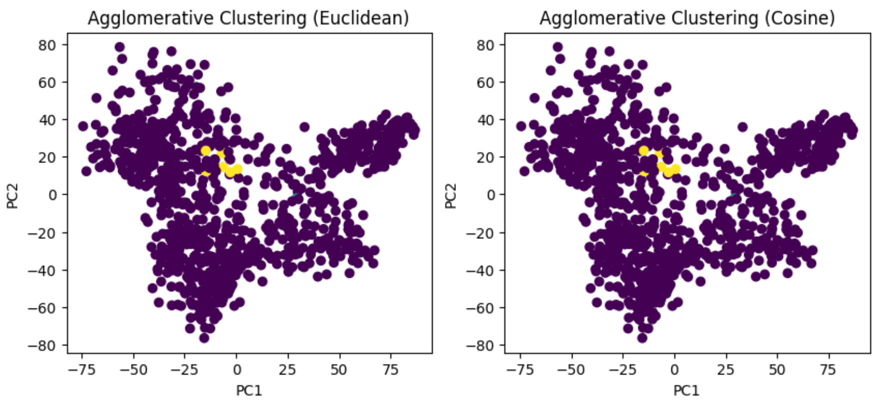

Figure 9: PCA reduction of the non-normalized data with clusters identified by agglomerative clustering with average linkage.

Demonstrated thoroughly, when the feature data is not scaled, the clusters produced by the average linkage algorithm are identical regardless of the distance metric used.

B Regression (Non-Normalized Data)

When the algorithm was run on non-normalized feature data, the resulting warning was outputted: *"ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation"*.

Note that when the algorithm was run on normalized feature data, no such warning was outputted.

Regularization Analysis

An analysis was done to examine how the value of alpha affects the number of features selected by the LASSO regression model when the feature data was not normalized. This was implemented by iterating over the alpha values and recording the number of non-zero coefficients in the weight vector. The results of this analysis are shown in Figure 10.

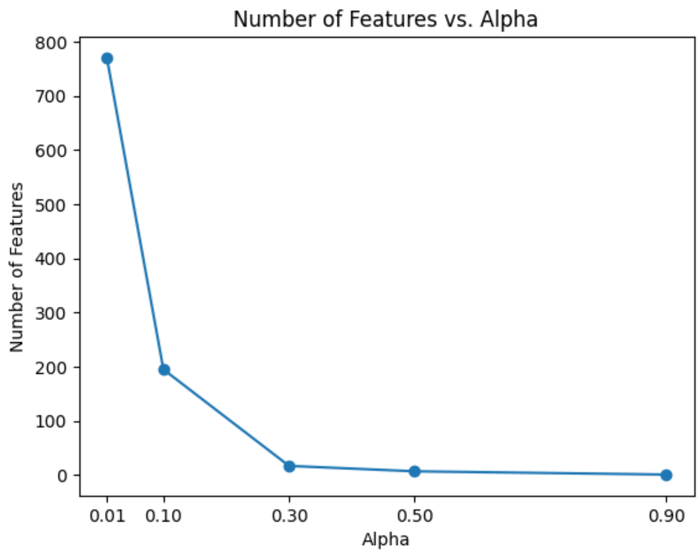

Figure 10: Number of non-zero coefficients in the weight vector as a function of alpha.

The results are also shown in Table 11.

Alpha	**Number of Non-Zero Coefficients**
0.01	770
0.1	195
0.3	17
0.5	7
0.9	1

Table 11: Number of non-zero coefficients in the weight vector as a function of alpha.

As expected, as alpha increases, the sparsity of the solution increases dramatically. This is conventional with theory, as more weight is given to the L1 norm penalty and thus the optimal solution will be more sparse. With this dataset, it is seen that an alpha value of 0.9 results in a weight vector with only 1 non-zero coefficient.

Cross-Validation Results

Outer Fold	Alpha	Mean Squared Error	Spearman Correlation	p-value
1	0.3	2.459	0.280	2.67×10^{-5}
2	0.3	2.375	0.246	2.39×10^{-4}
3	0.3	3.169	0.228	6.97×10^{-4}
4	0.3	2.314	0.347	1×10^{-7}

Table 12: Results of the nested cross-validation process.

The ideal alpha value with non-normalized data was found to be 0.3. The Spearman Rank Correlation at each outer loop fold was found to be statistically significant at a significance level of less than 0.05. Thus, the average MSE and Spearman Rank Correlation of a LASSO regression model were found to be **2.579** and **0.275** respectively.

Compared to the normalized data, the MSE and Spearman Rank Correlation values are worse. This is likely due to the fact that the data was not normalized, and therefore the LASSO regression model was not able to converge to the optimal solution.

Comparison to Baseline

The LASSO model trained on non-normalized data was compared to the mean and median dummy models. The results of this comparison are shown in Table 13.

Model	MSE	Spearman Rank Correlation
Mean	2.780	NaN
Median	2.798	NaN
LASSO	2.579	0.275

Table 13: Comparison of models.

The LASSO regression model trained on non-normalized data was found to outperform both the mean and median dummy models in MSE and Spearman Rank Correlation.

Discussion

In this analysis, the significance of normalizing the feature matrix data was highlighted. Clearly, a more optimal model was achieved when the LASSO model was trained on normalized data since the model was able to converge and achieve a lower MSE and higher Spearman Rank Correlation.