Methods in Bioinformatics

MSc Bioinformatics
University of Crete

Student: Eleni Mourouzidou
07/2023

Project on gene expression data clustering

Abstract:
The aim of this study is to investigate the molecular networks underlying viral-triggered asthma exacerbations in children. By analyzing gene expression data obtained during acute exacerbations and the subsequent recovery phase, which occurs within 7 to 14 days, we seek to elucidate the mechanisms involved in these exacerbations. Additionally, the construction of coexpression networks will contribute to a better understanding of the molecular mechanisms underlying asthma exacerbations and aid in the development of improved treatment strategies. Clustering algorithms will be employed to group genes with similar expression patterns in order to identify co-regulated gene clusters and gain insights into the biological pathways involved in asthma exacerbations.

Data obtained from:
DOI: 10.1016/j.jaci.2011.10.038
DataSet Record: GDS4424

Data extraction and cleaning
The data extraction and cleaning process for this project was performed using R language and the package “GEOquery”. Initially, the dataset of interest, identified as "GDS4424," was obtained using the getGEO() function. The expression data table was stored in a dataframe called data_df. To ensure data quality, missing values were assessed in the dataset. Rows that contained only missing values, indicating empty genes entries, were identified and removed from the data_df dataframe using the complete.cases() function. Additionally, a check for missing values per sample was performed using the is.na() function and the colSums() function, allowing for the determination of the number of missing values in each sample. By following these steps, the data extraction and cleaning process successfully ensured the integrity of the dataset, providing a reliable foundation for subsequent analyses and interpretation. The data extraction and cleaning process was executed successfully, resulting in a clean dataframe that does not contain any missing values. Every gene in the dataset has complete data information for all the samples. After the successful cleaning of the dataframe, the clean dataframe was saved to a TSV file called "expression_matrix.tsv" for further analysis in Python.

```
if (!require("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("GEOquery")
library(GEOquery)
eSet <- getGEO("GDS4424",
               destdir = '.',
               getGPL = F)

data_df <- eSet@dataTable@table
missing_rows <- !complete.cases(data_df)
data_df <- data_df[!missing_rows, ]
col_missing_count <- colSums(is.na(data_df))
print(col_missing_count) # outputs 0
write.table(data_matrix[, -2], file = "expression_matrix.tsv", sep
= "\t", quote = FALSE, row.names = FALSE)
```

Data exploration and preprocessing

The "expression_matrix.tsv" file was loaded, for further analysis in Python, into a DataFrame using the get_data() function. This file contained the gene expression data obtained from the previous cleaning process. The resulting DataFrame, expression_df, consisted of 32 columns representing the samples and 32,464 rows representing each gene's expression levels. To gain insights into the dataset, the sample labels were assigned based on the pattern of "post infection, during exacerbation." The labels were represented using a NumPy array called labels, with the value 1 indicating the "after infection" condition and 0 representing the "during exacerbation" condition. This encoding enabled the distinction between the two conditions. To incorporate the labels into the DataFrame, a new row named 'Condition' is added, and the labels are assigned to this row using the loc function. Finally, the modified DataFrame is returned as the output of the load_expression_data() function.

```
def get_data(filepath):
    return pd.read_csv(filepath, delimiter="\t")


def load_expression_data(filepath):
    expression_df = get_data(filepath)

    # store sample labels according to the pattern post infection,
```

```
during exacerbation
    # 0 representing during exacerbation condition and
    # 1 representing after infection condition
    labels = np.array([1, 0] * (expression_df.shape[1] // 2))
    expression_df.loc['Condition'] = labels

    return expression_df
```

In order to gain insights from the gene expression data, the get_data_insights() function was implemented. This function takes the expression_df DataFrame as input and performs calculations of the min, max, range and median values.

```
def get_data_insights(expression_df):
    # Get the minimum and maximum values of the DataFrame
    # to adjust the scale of visualization later
    min_value = expression_df.min().min()
    max_value = expression_df.max().max()
    value_range = max_value - min_value
    median_value = expression_df.median()

    return min_value, max_value, value_range, median_value
```

To achieve a better clustering visualization and analysis of the gene expression data based on different conditions, the conditions_grouped_df() function was implemented. This function takes the expression_df DataFrame as input and groups the data by conditions. Firstly, the columns in expression_df that correspond to condition 0 (during exacerbation) and condition 1 (after infection) were identified using boolean indexing. The resulting columns were stored in separate variables, columns_0 and columns_1, respectively. Finally, a new DataFrame was created by concatenating two subsets of expression_df: expression_df[columns_0] and expression_df[columns_1] in order to ensure that columns with the same condition are grouped together in the new Data Frame.

```
def conditions_grouped_df(expression_df):
    # Store the columns with label == 0 and label == 1 separately
    columns_1 = expression_df.columns[expression_df.iloc[-1] == 1]
    columns_0 = expression_df.columns[expression_df.iloc[-1] == 0]
    return pd.concat([expression_df[columns_0],
expression_df[columns_1]], axis=1)
```

To focus on genes that exhibit higher expression changes, the filtered_var_df() function was implemented to filter the gene expression data based on variance. This function takes the expression_df DataFrame and a threshold value as inputs. Next, the function stores the variance for each gene in a new Data Frame.

In conclusion, the filtered_df DataFrame contains the gene expression data filtered based on the variance threshold, while the range_df DataFrame holds the variance values for each gene. These filtered and variance-related data will be further utilized to observe the changes of the clustering model's performance according to the threshold of variance.

```
def filtered_var_df(expression_df, threshold):
    # Specify the variance of gene expression levels -
    # remove genes with very low variance
    onlyexp_df = expression_df.drop('Condition')
    range_values = onlyexp_df.var(axis=1)
    range_df = pd.DataFrame(range_values, columns=['Range'])
    filtered_df = onlyexp_df[range_values >= threshold]

    return filtered_df, range_df
```

Clustering with K-means

For this project, the K-means clustering method was employed. The perform_kmeans_clustering() function utilized the KMeans class from scikit-learn to perform the clustering and was developed to apply this algorithm to the gene expression data. K-means clustering approach, assigns data points to clusters based on their proximity to cluster centroids. Here, we used clusters = 2 to investigate the two conditions: during exacerbation and after infection based on our bias. In summary, the perform_kmeans_clustering() function returns

three outputs: the original data DataFrame with the cluster labels, the data_sorted DataFrame for enhanced visualization, and the cluster_labels array that stores the cluster assignments for each data point.

```
def perform_kmeans_clustering(data, num_clusters=2):
    k_means = KMeans(num_clusters)
    k_means.fit(data)
    cluster_labels = k_means.labels_
    data['Cluster'] = cluster_labels
    data_sorted = data.sort_values(['Cluster'])
    return data, data_sorted, cluster_labels
```

In order to check the quality of our clustering results and determine the optimal variance threshold for grouping the gene expression data, two functions were implemented: performance() and plot_silhouette_scores(). The performance() function calculates the silhouette score, which serves as a measure of cluster separation, based on the provided data and labels.

The plot_silhouette_scores() function visualizes the silhouette scores, using MatplotLib for different variance thresholds. It iterates over a range of thresholds and performs the following steps: grouping the gene expression data based on conditions using conditions_grouped_df(), filtering the data based on the variance threshold using filtered_var_df(), performing K-means clustering on the filtered data using perform_kmeans_clustering(), and calculating the silhouette score for the clustered data using performance(). The silhouette scores for each threshold are stored and plotted against the variance thresholds.

This analysis aids in understanding the impact of different variance thresholds on the clustering performance and assists in determining the most appropriate threshold for further analysis and interpretation of the gene expression data.

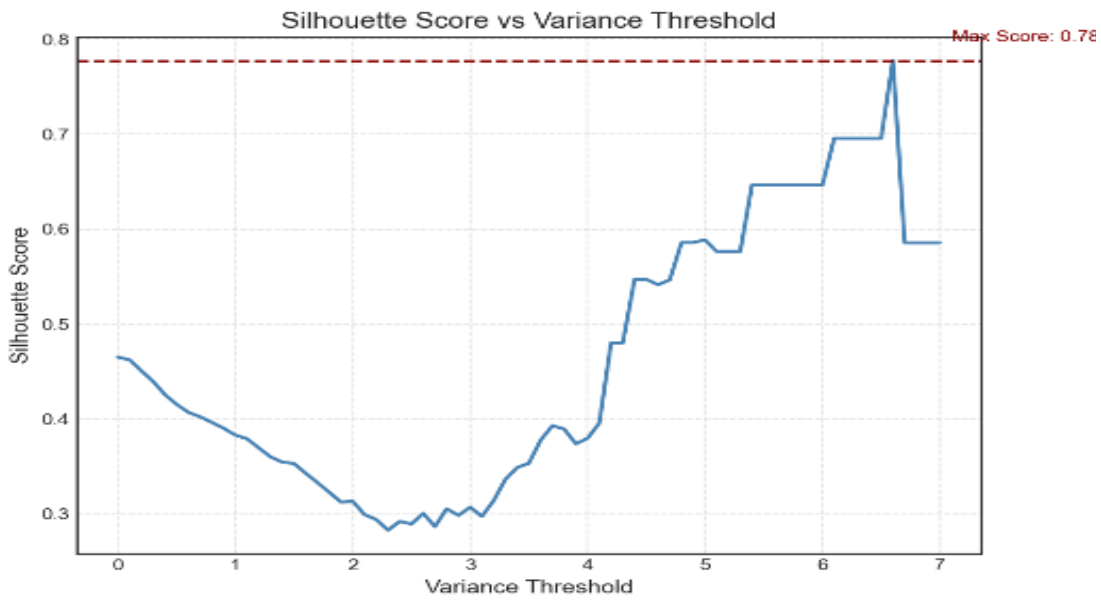

Figure 1. Relationship between the variance thresholds and the resulting Silhouette Score

Based on these results, we chose to analyze further and proceed to a cluster analysis for three different variance thresholds:

For a variance threshold of 0, the silhouette score is 0.466.
For a variance threshold of 5, the silhouette score is 0.589.
For a variance threshold of 6.6 (the threshold corresponding to the maximum silhouette score), the silhouette score is 0.778.

```
def perform_kmeans_clustering(data, num_clusters=2):
    k_means = KMeans(num_clusters)
    k_means.fit(data)
    cluster_labels = k_means.labels_
    data['Cluster'] = cluster_labels
    data_sorted = data.sort_values(['Cluster'])
    return data, data_sorted, cluster_labels


def performance(data, labels):
    return silhouette_score(data, labels)
```

```
def plot_silhouette_scores(expression_df, min_threshold,
max_threshold, step_size):
    silhouette_scores = []
    thresholds = np.arange(min_threshold, max_threshold +
step_size, step_size)

    for threshold in thresholds:
        transformed_df = conditions_grouped_df(expression_df)
        filtered_df, range_df = filtered_var_df(transformed_df,
threshold)
        clustered_data, clustered_data_sorted, cluster_labels =
perform_kmeans_clustering(filtered_df)
        silhouette = performance(clustered_data, cluster_labels)
        silhouette_scores.append(silhouette)

    plt.figure(figsize=(8, 6))
    plt.style.use('seaborn-white')
    ax = plt.gca()
    ax.set_facecolor('white')
    plt.plot(thresholds, silhouette_scores, color='steelblue',
linewidth=2)

    plt.xlabel('Variance Threshold', fontsize=12)
    plt.ylabel('Silhouette Score', fontsize=12)
    plt.title('Silhouette Score vs Variance Threshold',
fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.5)
    max_score = max(silhouette_scores)
    plt.axhline(y=max_score, color='darkred', linestyle='--')
    plt.text(max_threshold + 0.1, max_score + 0.02, f"Max Score:
{max_score:.2f}", color='darkred', fontsize=10)

    plt.show()
```

Clustering Visualization

To visualize the gene expression patterns, a function that generates a heatmap representing different gene expression values among different conditions was implemented. For a better visualization, the range of colors on the colormap corresponds to the range of expression values in the data.

The title of the heatmap includes the variance threshold that was used to filter the data, as we are going to use this to compare the changes of the model performance for different variances.

In figure 2, the original dataset is clustered (as the variance threshold = 0). The samples are sorted by condition (during and after infection), having on the left the condition 0 representing the exacerbation state, and on the right the condition 1 representing the state after 1-2 weeks passed from the exacerbation. The y-axis represents the gene clusters, as the genes have been clustered and sorted vertically. Genes with high expression values are represented by yellowish colors, while genes with low expression values are represented by bluish colors.
Analyzing the heatmap, we can observe that in the upper left part (one cluster), there are higher expression values indicated by yellowish colors, while the lower right part (the other cluster) consists of genes with lower expression values indicated by bluish colors. This suggests that the two clusters exhibit different expression patterns.

Additionally, when comparing the two conditions along the x-axis, there are no significant differences in gene expression patterns. This observation indicates that the gene expression patterns remain relatively stable across the two conditions, as both clusters exhibit similar expression patterns.

In the right part of the heatmap, represented by bluish colors, there are some lines indicating genes with lower expression values. On the other hand, in the left part of the heatmap, we can hardly see some blue lines, indicating a higher prevalence of genes with higher expression values.
These differences in the distribution of blue lines suggest that certain genes exhibit different expression patterns between the conditions. The presence of blue lines in the right part indicates a subset of genes with lower expression levels specific to that condition. This observation suggests that these genes may play a role in distinguishing or characterizing the condition represented on the right side of the heatmap.

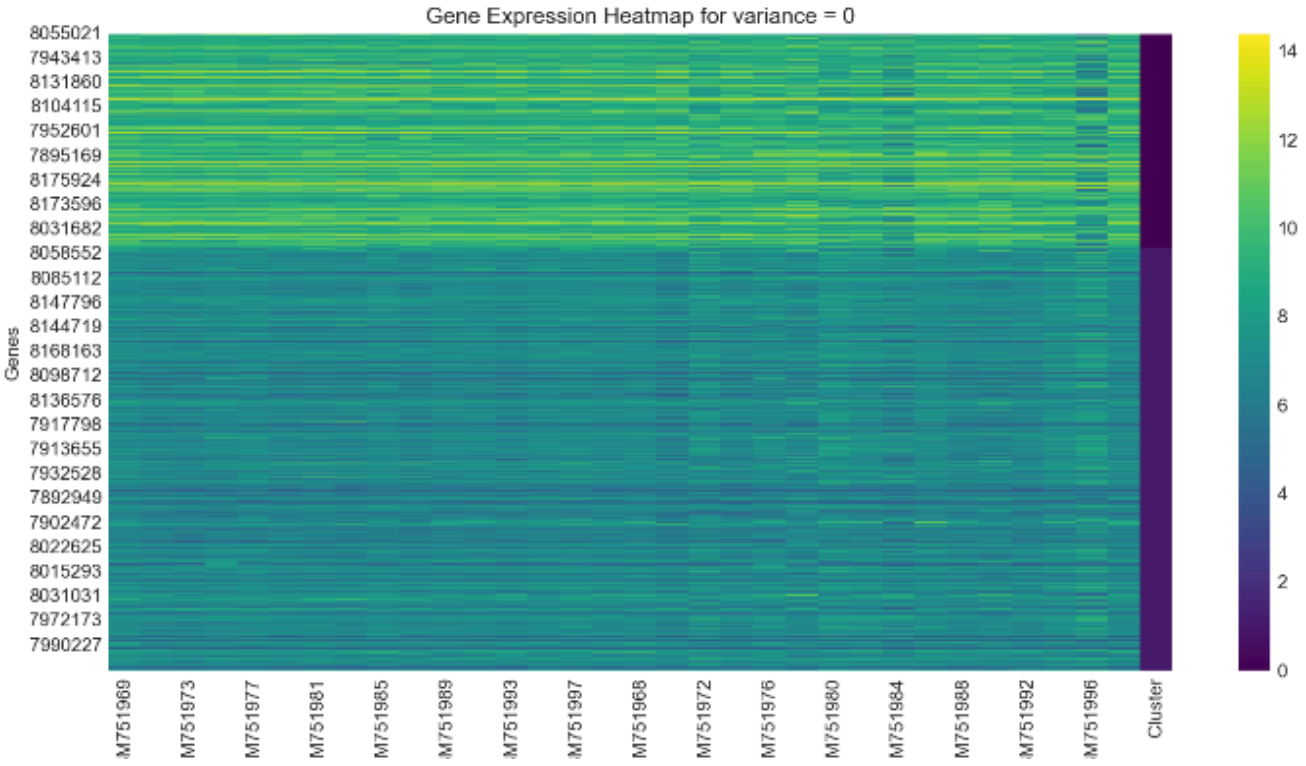

Figure 2. Gene expression heatmap for variance = 0 (original dataset)

In the heatmaps of figure3, and 4 the genes belong to the same initial cluster. Thus, there is not much to compare on the y axis, but observing the x axis ϛe can clearly see that the highest expression levels are distributed to the bottom left part where the samples belong to the condition of exacerbation and on the right part, the genes are characterized by lower expressions, as the samples belong to the after infection condition.

The high expression levels in the left part indicate that these genes are upregulated during the infection condition, while the low expression levels in the right part suggest downregulation or very low expression 1-2 weeks after the infection. The genes in this cluster that are highly related to the infection state may play significant roles in the immune response or other biological processes associated with the exacerbation condition.

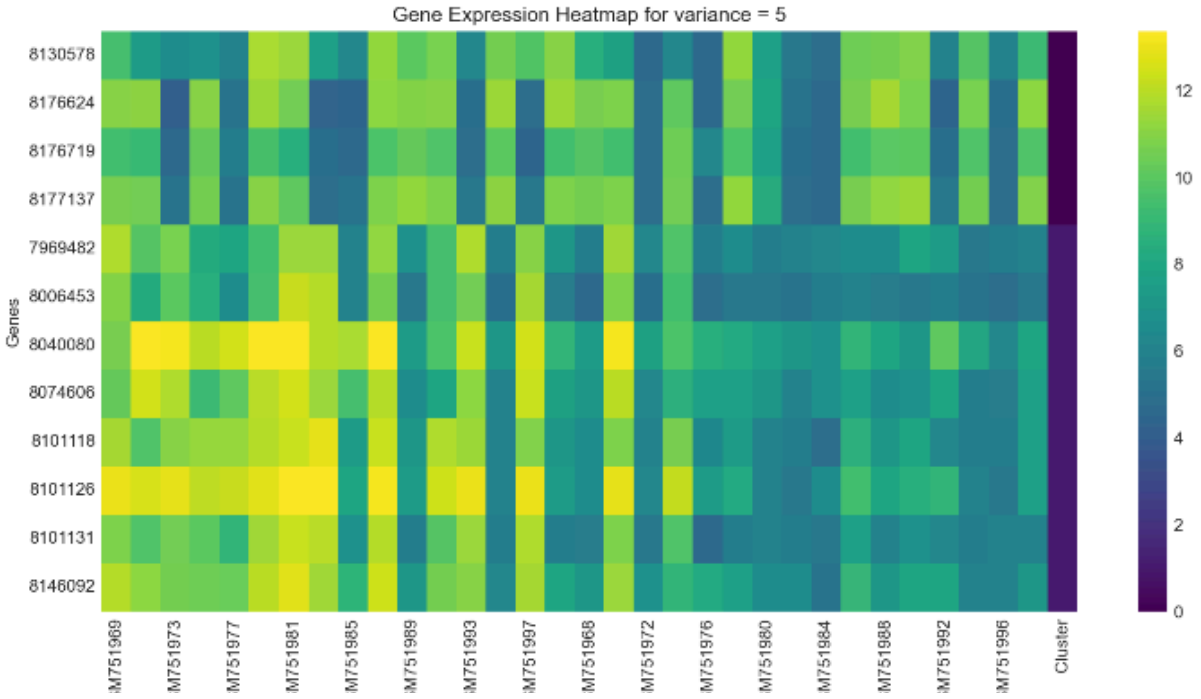

Figure 3. Gene expression heatmap for variance = 5

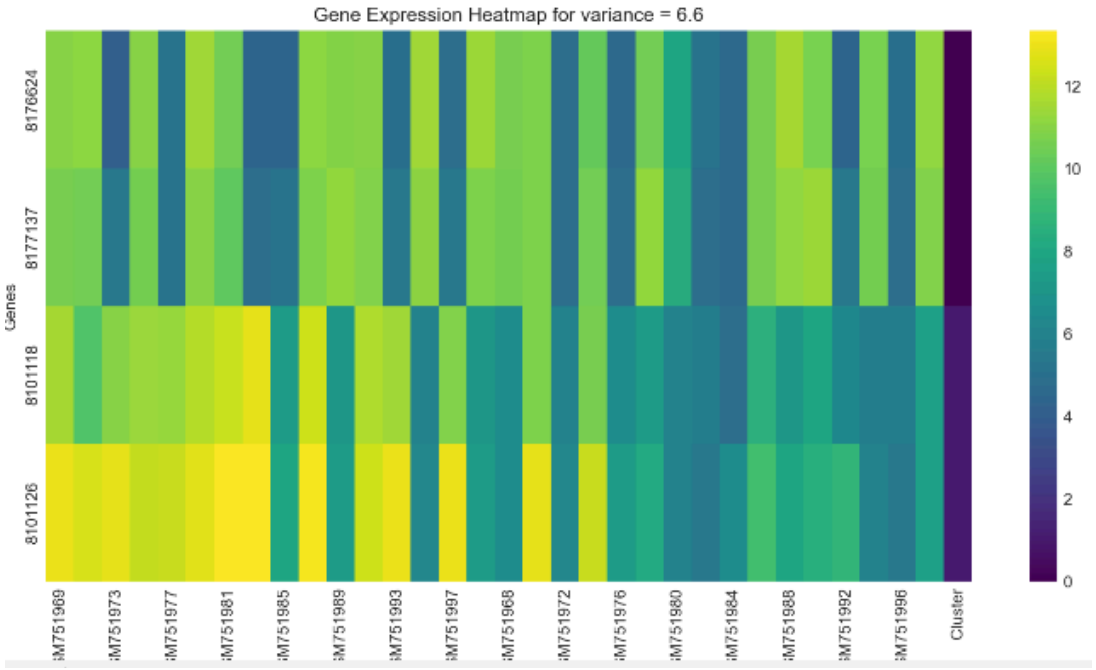

Figure 4. Gene expression heatmap for variance = 6.6 (for maximum silhouette score)