Machine Learning and Big Data for Bioinformatics

Module II - Non-Supervised Learning: Clustering and Association Rules

Capsule 2 - Clustering

Authors

Carlos Cano Gutiérrez

Associate Professor at University of Granada, Department of Computer Science and ICT.

INDEX

In this notebook:

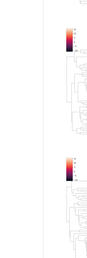
1. You will learn how to apply different clustering algorithms theoretical and in practice to identify groups of genes and samples in gene expression matrices
2. You will learn how to interpret the results of clustering algorithms. You will visually evaluate the quality of the results obtained.
3. You will learn how to use hierarchical clustering to evaluate the results of clustering models and choose the number of groups (K) that best capture the structure of the data.

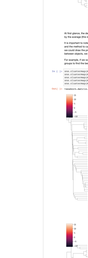
Contents:

1. Introduction to Microarrays
2. Hierarchical Clustering
3. K-means and K-medoids Clusters
4. Additional Content

INSTALLATION OF LIBRARIES

In [1]:
```
# Cell install dependencies
# Celda requirements
!pip install -q -U numpy
!pip install -q -U pandas
!pip install -q -U matplotlib
!pip install -q -U seaborn
!pip install -q -U scipy
!pip install -q -U scikit-learn
!pip install -q -U kneed
!pip install -q -U ipywidgets
!pip install -q -U ipykernel
!pip install -q -U ipykernel-core
!pip install -q -U ipykernel-helper
!pip install -q -U ipykernel-data
!pip install -q -U ipykernel-tornado
!pip install -q -U ipykernel-matplotlib
!pip install -q -U ipykernel-ipyparallel
!pip install -q -U ipykernel-ipywidgets
!pip install -q -U ipykernel-nbformat
!pip install -q -U ipykernel-nbconvert
!pip install -q -U ipykernel-jupyterlab
!pip install -q -U ipykernel-jupyterlab-server
!pip install -q -U ipykernel-jupyterlab-widgets
!pip install -q -U ipykernel-widgetsnbextension
!pip install -q -U ipykernel-widgetsnbextension-3.6.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0
!pip install -q -U ipykernel-widgetsnbextension-3.6.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-0.0.0-
```





most popular ones are the Silhouette Index (`silhouette`) and the Calinski-Harabasz Index (`calinski_harabasz`).

The following code illustrates how to calculate the value of these indices for different values of `k`. The value of `k` that obtains higher values for these indices will be the chosen value. The `plotkmeansfitsscores(mkmeansfitsscore)` function allows you to do this whole process with a line fitter, graphically representing the values of the indices obtained for different `k`.

In [ ]: from **pelican_kit.cluster** import KSlineFitsScore

```
data = df[df.columns]

model = KMeans()
visualizer = KSlineFitsScore(model, k=(1,11), metric='silhouette', timings=False, locate_elbow=True)
visualizer.fit(data)
visualizer.show()
```

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>

For example, according to the silhouette index, the optimal value of `k` is 2. We now try the Calinski-Harabasz metric:

In [ ]: visualizer = KSlineFitsScore(model, k=(1,11), metric='calinski_harabasz', timings=False, locate_elbow=True)
visualizer.fit(data)
visualizer.show()

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b1d8a90>

With Calinski-Harabasz the optimal value of `k` is 2.

If we want to calculate the goodness of clustering on a particular result with the Silhouette or Calinski-Harabasz indices, we would proceed as shown in the following cell of code:

In [ ]: from **sklearn** import Kmeans

```
# 1- Generamos el clustering
k_means = KMeans(n_clusters=2)
model = k_means.fit(df.transpose())
cluster_labels = k_means.predict(df.transpose())

# 2- Lo evaluamos con el coeficiente silhouette (silhouette low indicates clusters is fallsely-Harabasz pick one result
from
sklearn.metrics import silhouette_score,metrics, cluster_predict
print("Silhouette Coefficient:%0.3f",%
% metrics.silhouette_score(df.transpose(), cluster_predict)
print("Indice Calinski-Harabasz: %f",%(metrics.CH(
```

```
Indice Silhouette: 0.1.Coefficient:0.3f%(0.598)
Indice Calinski-Harabasz: 2.79
```

3.3. K-means on full expression matrices

With what we learned above, we apply K-means on a full expression matrix. To do so, we first estimate the optimal number of `k`. In these estimates, we turn on the `timings=True` option to also have a graph that contains on the graph with the time consumed by the clustering for each value of `k` that is tested:

In [ ]: from **pelican_kit.cluster** import KSlineFitsScore

```
data = df[df.columns]

model = KMeans()

visualizer = KSlineFitsScore(model, k=(1,11), metric='silhouette', timings=True, locate_elbow=True)
visualizer.fit(data)
visualizer.show()
```

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>

```
visualizer = KSlineFitsScore(model, k=(1,11), metric='calinski_harabasz', timings=True, locate_elbow=True)
visualizer.fit(data)
visualizer.show()
```

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>

From the view of both plots, it appears that `k=2` is the result most faithful to the structure and variability of the underlying data. We apply k-means for `k=2` and plot the resulting clusters:

In [ ]: k_means = KMeans(n_clusters=2)
model = k_means.fit(df.transpose())
cluster_labels = k_means.predict(df.transpose())
clusters = pd.DataFrame(cluster_predict,index=data.index,columns=['cluster'])

```
print("Tamaño de cada clusters:")
size=clusters.groupby('cluster').size(_inverse)
print(size.sort_values(ascending=False))
# - global loc the (10.1011) * sum_c(10.1011*centroidsmean))
```

```
centers = pd.DataFrame(k_means.cluster_centers_,columns=list(data))
centers.stack().to_csv(

Tamaño de cada clusters:
0:  442 (20.20%)
1:  1.  (0.05%)
```

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0ee4f84a8b>

4. ADDITIONAL CONTENT

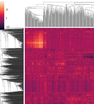
4.1. Outlier detection and removal

Function `yellowbrick` allows to identify and omit outliers from the result representation.

In [ ]: # yellowbrick module
pip install yellowbrick, matplotlib

```
# yellowbrick/yellowbrick-develop/dc de yellowbrick
pip install yellowbrick,matplotlib
```

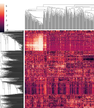
Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>



4.2. CHANGING THE PLOTS

In [ ]: # Cambiar en la paleta de colores
pip install yellowbrick,matplotlib, 'seaborn', n_colors=7, method='ward', cmap='Reds')
pip install yellowbrick,matplotlib, 'seaborn', n_colors=7, method='ward', cmap='viridis')
pip install yellowbrick,matplotlib, 'seaborn', n_colors=7, method='ward', cmap='Blues')

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>



4.3. Data normalization

The `sklearn.preprocessing` function allows transformations to be performed on the input data.

- standardization: transforms the data so that it has mean 0 and variance 1. To apply this transformation to the rows, `standard_scaler()` to apply it to the columns.
- normalization: by calculating a norm parameter `p_scaler()` to apply this transformation to the rows, `p_scaler()` to apply it to the columns.

Thus, in case our starting data are not normalized, these options can be used for do it in the same `yellowbrick` function. In our case, we show these options to illustrate their effects on the data, although the normalization of the expression matrix has already been carried out in Module 0, so we do not need to apply further normalization measures.

In [ ]: # Normalizacion
pip install yellowbrick,matplotlib, standard_scaler()
Normalizacion
pip install yellowbrick,matplotlib, p_scaler()

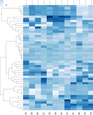
Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b5f5f90>

4.4. Hierarchical clustering on rows or columns only

To perform hierarchical clustering only on rows (genes), we need to use the following parameters

In [ ]: pip clusteringmap(mmmpp_rows, row_cluster=True, col_cluster=False)

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>



To perform only the cluster on the columns

In [ ]: pip clusteringmap(mmmpp_rows, row_cluster=False, col_cluster=True)

Out [ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1a8b4d5f90>

4.5. What Now? More clustering algorithms

There are numerous clustering algorithms already implemented in Python. After finishing this module, we recommend you to explore the tutorials in Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12: pp. 2825-2830, 2011. In particular, the chapter on Clustering (https://scikit-learn.org/stable/modules/clustering.html#clustering)

BIBLIOGRAPHY

- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12: pp. 2825-2830, 2011. En particular, el capítulo de Clustering (https://scikit-learn.org/stable/modules/clustering.html#clustering)
- Seaborn Clustermap (https://seaborn.pydata.org/generated/seaborn.clustermap.html)
- Cancer Genome Atlas Network. Comprehensive Genomic Characterization of Colorectal Adenocarcinoma. Cell. 2012;151(7):1881-1895. doi:10.1016/j.cell.2012.09.044 (https://www.cell.com/fulltext/S0092-8674(12)01483-X)

In [ ]: