# Machine Learning y Big Data para la Bioinformática

## Módulo 8 - Aprendizaje No Supervisado: Clustering y Reglas de Asociación

## Cápsula 2 - Clustering

**Autores**

**Cristina Gallego Gallego**

Profesor Titular de la Universidad de Granada, Departamento de Ciencias de Computación e Inteligencia Artificial.

**Mercedarias Inmaculada Lo Bosdevole**

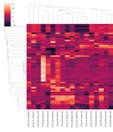
Alumna de doble cuatrimo (Profesional), de la grado para cuatro meses de ensayo de un concurso de obras hacer el análisis descriptivo y de un contrato.

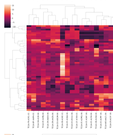
Este cuaderno es parte de la cápsula de aprendizaje no supervisado del curso de Machine Learning y Big Data para la Bioinformática. Aquí vamos a ver cómo se implementan algunos algoritmos de clustering (jerárquico y k-medias) para identificar grupos de pacientes con diferentes tipos de cáncer. Por otro lado, vamos a ver cómo se implementan una serie de métricas de distancia entre pacientes para comprobar las hipótesis descubiertas como las diferentes tipos de cáncer. Por último, vamos a ver cómo se implementan algunas técnicas de visualización para representarlos. En cada uno de estos algoritmos, vamos a ver cómo afectan a los resultados los diferentes parámetros de entrada y el principio en cada uno de ellos.

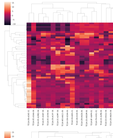
Antes de nada:

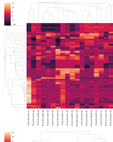
En los Ficy Hospitales que al comienzo de adolescencia "Ref-ref-ref-ref-dropear" dónde tardas, antes y la siguiente. En cada contrato, no permitirá ejecutar ningún algoritmo de machine learning hasta que no haya terminado de hacer el análisis descriptivo y de contrastar las hipótesis nulas. Cada uno de los cuadernos de la 01 a la cuarta cápsula - "Ok, ok, ok,

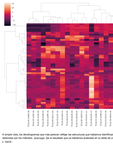
Salida(s):
```
Dendrograma_Hashtags_Clasificacion.png
```

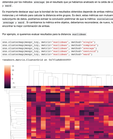












A simple vista, los dendrogramas que mejor separan clústeres de palabras que aparecen frecuentemente identificadas de manera conjunta (en los hashtags) son los obtenidos por los métodos `average` y `complete`, por el hecho de que la distancia entre grupos es el valor que utiliza para unirlos (`average`) o el máximo (`complete`).

Es importante destacar que aun la formación de los clústeres dependa de ciertas métricas, los dendrogramas para cada método de linkage tienden a ser similares pues coincide la distancia entre grupos. Por tanto, estima mirar con mayorías dependiendo. En este caso, del método de linkage `average` y `complete` se observan 3 clústeres bien definidos, mientras que en el caso de `single` y `ward` se observan 2 clústeres. Por último, el método `average` y `complete` se diferencian en que `average` es sensible a outliers, mientras que `complete` es más robusto. Como los clústeres de colores de dichos métodos son grupos, pues parecen ser más robustos.

Por ejemplo, la palabra social aparecía también para dichos clústeres `social:blue`.

```
In [4]: df[df[df[
```

In [14]:
```
k_mean = KMeans(n_clusters=2)
clusters = k_mean.fit(X_scaled)
cluster_predict = k_mean.fit_predict(X_scaled)
cluster_centers = k_mean.cluster_centers_

print("Centros de cada cluster:")
print(cluster_centers[:,:2]*data_std+
data_mean[:2]*data_std+
    print("\n\n 150 116 201\n\n 0 154.1,1.1777\n\n\n\n\n\n")

centroid = pd.DataFrame(k_mean.cluster_centers_,columns=list(data.iloc[:,:2]))
centroid.T

Centros de cada cluster:
0   -0.0 -0.0
1   0.0 0.0
dtype: float64
```

Out[14]:

Probablemente obtengamos ejemplos de resultados carentes con cada aplicación un cluster mayoritario del 80% de elementos y un outlier que conforma su propio grupo. Esto solucíon no resulta demasiado informativa y lejos de ser óptima a concepar que hay una muestra con comportamientos muy diferentes al resto.

De esta moda sentar a priori el número de grupos de los datos es una tarea muy relevante y determinadl en gran medida los resultados obtenidos por el algoritmo clustering.

Afortunadamente hay ocasiones para saber cómo determinar el número óptimo de clusters para ciertos conjuntos de **X** y evitar comportamientos no resultados obtenidos, elegiendo aquellos que obtienen el mejor comportamiento de los datos. Para determinar el mejor resultado, se utilizan diferentes métricas de bondad del agrupamiento. Las más usuales son el índice Silueta (`silhouette`) y el índice de Calinski Harabasz (`calinski_harabasz`).

El siguiente código traza dichos índices calculados a partir de datos indicados para distintos valores de **K**. El valor de **K** que obtenga valores más altos para estos índices será el que nos indique el número de clusters que mejor se ajusta a los datos. Por ejemplo, `silhouette` permite hacer todo esto y además con una sola curva Traza, representando gráficamente los valores de los índices obtenidos para los distintos **K**.

In [15]:
```
from yellowbrick.cluster import SilhouetteVisualizer

data = mergo_log.iloc[:,:2]
X = data.values
model = KMeans(n_clusters=2, random_state=42, n_jobs=-1, verbose=0)
visualizer = SilhouetteVisualizer(model, colors='yellowbrick')
visualizer.fit(X)
visualizer.poof()
```

Out[15]:

Hay ejemplos, según el índice silueta, el valor óptimo de **K** es 2. Probemos ahora con la métrica Calinski Harabasz:

In [16]:
```
visualizer = SilhouetteVisualizer(model, n_jobs=-1, metric='calinski_harabasz', colors=[#ffa54c,#ffa54c])
visualizer.fit(X)
visualizer.poof()
```

Out[16]:

Con Calinski Harabasz el valor óptimo de **K** es 3.

El mencionado calcular la bondad del clustering sobre un resultado conviene con los índices Silueta y Calinski Harabasz, procediéndamos como no muestra en la siguiente celda de código:

In [17]:
```
from math import floor

# 2 iteraciones
kmax = 10
for k in range(2,kmax+1):
    k_means = KMeans(n_clusters=k)
    clusters_predict = k_means.fit_predict(data)
    print("\n\n",
    "Para n_clusters =",k,"el coeficiente silueta es :",
    "\n",
    silhouette_score(data, clusters_predict, metric='euclidean',
    sample_size=floor(0.1*len(data)),
    random_state=123),
    "\n",
    "Para n_clusters =",k,"el coeficiente Calinski-Harabaz es :",
    "\n",
    calinski_harabaz_score(data, clusters_predict),
    "\n\n\n\n\n\n")

Para n_clusters = 2 el coeficiente silueta es : 0.5976999999999999

 Para n_clusters = 2 el coeficiente Calinski-Harabaz es : 4.179
```

### 3.3. K-means en matrices de expresión completas

Con lo aprendido anteriormente, aplicamos K-medias en una matriz de expresión completa. Para ello, primero extraemos el número óptimo **K**. En estas ventiladores aplicamos la opción `silhouette`, para luego que también sea útil en gráficos una línea vertic con el número que consuma el clustering por cada valor de **K** que se ensaya.

In [18]:
```
from yellowbrick.cluster import SilhouetteVisualizer

data = mergo_log[:,:]
X = data.values

model = KMeans()
visualizer = SilhouetteVisualizer(model, n_jobs=-1, metric='silhouette', colors=[#ffa54c,#ffa54c])
visualizer.fit(X)
visualizer.poof()
```

Out[18]:

In [19]:
```
visualizer = SilhouetteVisualizer(model, n_jobs=-1, metric='calinski_harabasz', colors=[#ffa54c,#ffa54c])
visualizer.fit(X)
visualizer.poof()
```

Out[19]:

Ya la cría de ambas gráficas parece que **K** = 2 es el resultado más fiel a la estructura y cantidadad de los datos subyacentes. Aplicamos k-medias para **K** = 2 y probemos los centroides resultantes:

In [20]:
```
k_means = KMeans(n_clusters=2)
clusters_predict = k_means.fit_predict(data)
centroid = pd.DataFrame(k_means.cluster_centers,columns=list(data.iloc[:,:2]))
centroid.T

Centros de cada cluster:
0   -0.0 -0.0
1   0.0 0.0
dtype: float64
```

Out[20]:

## 4. CONTENIDO ADICIONAL

### 4.1. Detección de outliers

La función `sns.clustermap` permite identificar y omitir `outliers` de la representación del resultado.

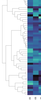
In [21]:
```
# Calculamos outliers
sns.clustermap(mergo, metric='euclidean', method='average')

# Ejemplos utilizando detección de outliers
sns.clustermap(mergo, metric='euclidean', method='average', robust=True)
```

Out[21]:

### 4.2. Cambios en la representación gráfica

In [22]:
```
# Cambiando los colores de el dendrograma
sns.clustermap(mergo_log, metric='euclidean', m_pixels=1, method='ward', cmap='hot')
sns.clustermap(mergo_log, metric='euclidean', m_pixels=1, method='ward', cmap='Pastel1')
sns.clustermap(mergo_log, metric='euclidean', m_pixels=1, method='ward', cmap='Pastel2')
```

Out[22]:



### 4.3. Normalización de los datos

La función `clustermap` permite llevar a cabo transformaciones sobre los datos de entrada:

- `standardize(axis)` del range de las variables (0, 1) dependiendo `standardize(axis)` para aplicar esta transformación a las filas, `standardize(axis=1)` para aplicarla a las columnas)
- normalización calculando z-score (poniendo `z_score(0)` para aplicar esta transformación a la filas, `z_score(1)` para aplicarla a las columnas)

De este modo, en caso de que nuestros datos de partida no estén normalizados, podrían ofrecerse estas opciones para hacerlo en la misma función `clustermap`. Por ejemplo, en nuestro caso, podríamos aplicar la opción `z_score(1)` para aplicar la normalización de datos tanto de expresión ya que los ficheros a utilizar en Bioinfo-S, al que no necesitamos aplicar dos medidas de normalización.

In [23]:
```
# Normalizaciones
sns.clustermap(mergo_log, standard_scale=1)
# Normalizaciones
sns.clustermap(mergo_log, z_score=1)
```

Out[23]:

### 4.4. Clustering pinípodos solo en filas o columnas

Para efectuar clustering pinípodos solo sobre las filas (genes), necesitamos utilizar los siguientes parámetros:

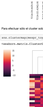
In [24]:
```
sns.clustermap(mergo_log, row_cluster=False, col_cluster=True)
```

Out[24]:

Para efectuar sólo el cluster sobre las columnas:

In [25]:
```
sns.clustermap(mergo_log, row_cluster=False, col_cluster=True)
```

Out[25]:



### 4.5. ¿Y ahora qué? Más algoritmos de clustering

Existen numerosos algoritmos de clustering en implementaciones de Python. Tras finalizar este módulo, sin recomendamos explorar las bondades de Scikit learn Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. En particular, el capítulo de Clustering: http://scikit-learn.sourceforge.net/stable/modules/clustering.html

## REFERENCIAS BIBLIOGRÁFICAS


- Scikit learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. En particular, el capítulo de Clustering: http://scikit-learn.sourceforge.net/stable/modules/clustering.html#clustering
- Yellowbrick: Visualizing the Scikit Learn machine learning pipeline. http://www.scikit-yb.org/en/latest/index.html
- Cancer Genome Atlas Network. Genomic Classification of Cutaneous Melanoma. Cell. 2015;161(7):1681-1696. doi:10.1016/j.cell.2015.08.044. https://www.ncbi.nlm.nih.gov/pubmed/26091043