Evaluación de Métodos de *Clustering*

Aprendizaje Automático II, 3º GRIA

Iria Agulló Gómez, 49682455V, iriaagullo22@gmail.com

Sara Porto Álvarez, 54153770V, saraportoalvarez@gmail.com

Introducción

En esta práctica se propone comparar diversos métodos de clustering, aplicándolos a un dataset común. Se estudiarán los algoritmos de K-Means, DBSCAN y Gaussian Mixtures, aplicados desde la librería de scikit-learn.

El dataset escogido ha sido el de Iris Dataset, que se conforma de datos de 150 flores iris de tres especies diferentes: Iris setosa, Iris versicolor e Iris virginica. Este conjunto de datos contiene información sobre atributos de la longitud y anchura del pétalo y el sépalo de estas flores, en centímetros; así como la especie a la que pertenecen.

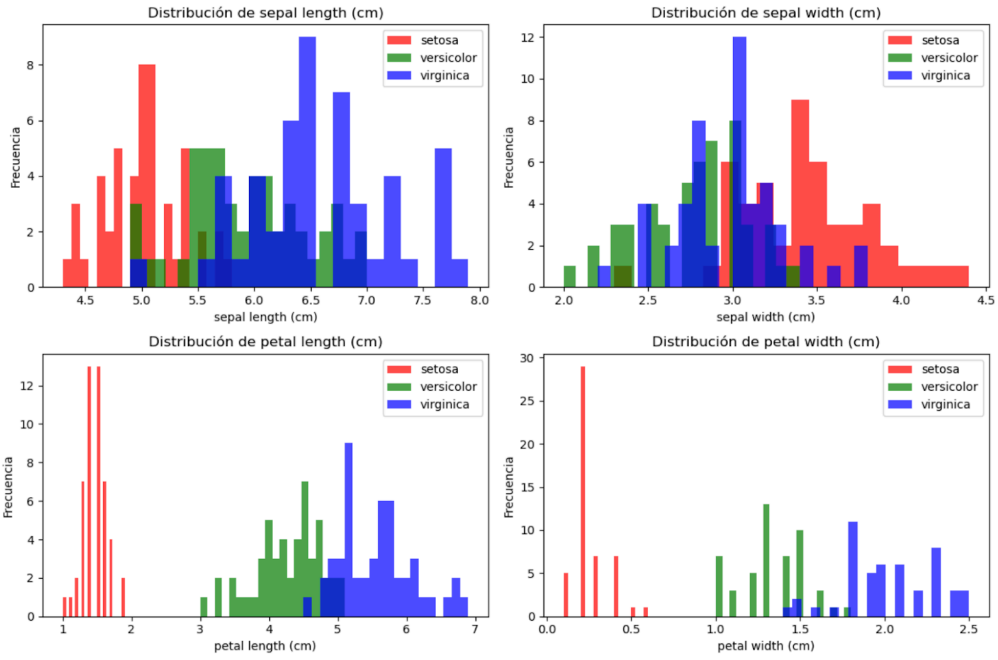
El objetivo será determinar cuál es el mejor método de clustering, de entre los tres estudiados, para diferenciar las tres clases de flor que hay en el Iris Dataset.

Descripción del proceso

Preparación y análisis exploratorio de los datos

El iris dataset se compone de los atributos de longitud y anchura del pétalo y del sépalo. Aunque, tras investigar sobre dichos atributos y ver que el menos significativo de ellos para hacer clustering era la anchura del sépalo, decidimos eliminar esta misma columna; dado que también sería más sencillo tener tres dimensiones a la hora de graficar y hacer una posterior evaluación.

A continuación se presenta una gráfica en la que se muestra la distribución de datos en cada uno de los atributos por cada una de las clases. Así, vemos que los atributos relacionados con el pétalo serán los más discriminativos; y los del sépalo los que más se superponen entre sí (siendo, como ya comentamos, la anchura del sépalo el atributo que menos información aportará). Otra conclusión que se puede sacar es que las clases que más difíciles serán de diferenciar serán la virginica y la versicolor.



Además, se muestran las primeras 5 instancias del dataset original:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

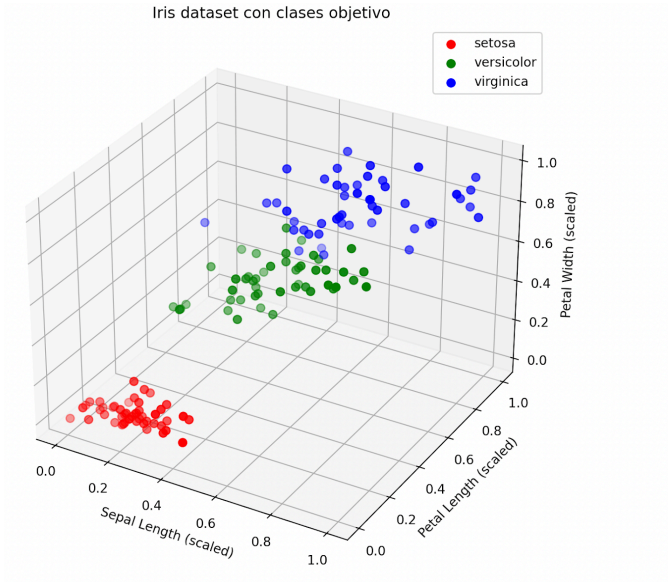
Por lo tanto, el primer paso del preprocesado de los datos será eliminar la columna de sepal width. Y, tras comprobar que no había datos nulos, aplicamos una normalización MinMaxScaler únicamente a las características del dataset (no al target, ya que daría errores de asignaciones de clases). Esto para que el clustering se vea lo menos afectado por las variaciones de rangos entre las distintas medidas de longitud.

Después de preprocesar los datos, las primeras 5 instancias del dataset se verían de la siguiente manera, y los datos serían de tipo flotante (las longitudes y anchuras) y entero (el target). Habría un total de 150 instancias, con 4 atributos. También se muestra una descripción estadística del dataset preprocesado.

	sepal length (cm)	petal length (cm)	petal width (cm)	target
0	0.222222	0.067797	0.041667	0
1	0.166667	0.067797	0.041667	0
2	0.111111	0.050847	0.041667	0
3	0.083333	0.084746	0.041667	0
4	0.194444	0.067797	0.041667	0

	sepal length (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000
mean	0.428704	0.467458	0.458056	1.000000
std	0.230018	0.299203	0.317599	0.819232
min	0.000000	0.000000	0.000000	0.000000
25%	0.222222	0.101695	0.083333	0.000000
50%	0.416667	0.567797	0.500000	1.000000
75%	0.583333	0.694915	0.708333	2.000000
max	1.000000	1.000000	1.000000	2.000000

La distribución de los datos en las tres dimensiones elegidas, con los datos preprocesados sería la que se muestra a continuación; por lo que se procurará que los clusters obtenidos con los algoritmos escogidos sea lo más similar a esta gráfica:



Entrenamiento de los modelos

Para el entrenamiento de los modelos se ha empleado la librería de scikit-learn. Lo primero que se hace es eliminar la columna de target para poder hacer el clustering únicamente con los atributos escogidos. Por lo que el número de clusters a buscar será igual al número de tipos de flores del dataset, es decir, tres.

Los algoritmos que hemos seleccionado para la realización del clustering son los que se mencionan a continuación.

K-Means

Este algoritmo divide los datos en k grupos, optimizando la distancia a los centroides. Se crea el modelo, con 3 clusters (el número de clases que hay) y una inicialización de los centroides apoyada en k-means++. Esto acelerará la convergencia y mejorará la calidad de los clusters en comparación con la inicialización aleatoria. Después, se ajusta el modelo con los datos.

```
## --- KMEANS CLUSTERING --- ##
kmeans = KMeans(n_clusters=3, init='k-means++') # inicialización con k-means++
kmeans.fit(df_scaled) # ajusta modelo
```

DBSCAN

El DBSCAN es un tipo de clustering basado en densidad; que agrupa datos según densidad, identificando ruido y formando clusters de forma arbitraria.

Para inicializar el clustering, se han de escoger valores para los hiperparámetros épsilon (distancia de vecindad, definida en 0.1) y minPts (mínimo de puntos necesarios para formar un cluster, que hemos fijado en 8). Los parámetros óptimos escogidos se seleccionaron después de probar distintos valores y combinaciones de épsilon y minPts. De nuevo, se ajusta el modelo con los datos.

```
## --- DBSCAN CLUSTERING --- ##
eps = .1 # dist vecindario
min_samples = 8 # min muestras cluster

dbscan = DBSCAN(eps=eps, min_samples=min_samples) # inicializa DBSCAN
dbscan.fit(df_scaled) # ajusta modelo
```

GaussianMixture

Se trata de un modelo probabilístico que asume que los datos provienen de varias distribuciones gaussianas, permitiendo clusters con formas elípticas y asignaciones de pertenencia parciales. Esta característica puede no ser estrictamente necesaria para la distribución de los datos en el Iris Dataset, pero no por ello el resultado de aplicar este tipo de clustering será negativo.

El modelo utilizará tres componentes gaussianas (es decir, tratará de identificar 3 grupos), y el parámetro covariance_type indica que cada componente tendrá su propia matriz de covarianza completa, permitiendo modelar relaciones complejas entre las variables.

```
## --- GAUSSIAN MIXTURE CLUSTERING --- ##
gauss_mixture = GaussianMixture(n_components=3, covariance_type='full') # inicializa Gaussian Mixture
gauss_mixture.fit(df_scaled) # ajusta modelo
```

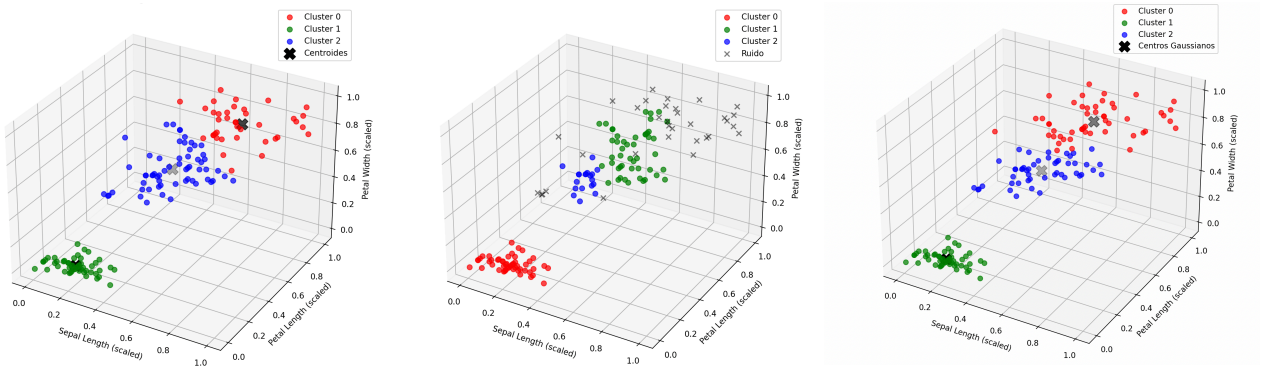
Análisis técnico

Con el objetivo de evaluar los distintos algoritmos de clustering, se extraen resultados de diversas métricas. Algunas de análisis interno del modelo (como son el coeficiente de Silhouette y el índice de Calinski-Harabasz), otras de análisis externo (el Adjusted Rand score) y también la gráfica de salida en 3D para sacar conclusiones sobre elementos más visuales.

Gráficas de matplotlib

Las tres gráficas obtenidas son las siguientes:

KMeans (3 clusters) **DBSCAN (3 clusters+ruido)** **GaussianMixture (3 componentes)**



A primera vista, observamos que el agrupamiento con DBSCAN produce datos con una cantidad excesiva de ruido; así como un cluster muy pequeño para la clase versicolor (azul). Parece que el cluster de clase virginica (en verde) se “come” puntos del grupo versicolor, y que la mayoría de puntos que deberían ser asignados al cluster se clasifican como ruido. Por otra parte, la clase setosa parece clasificarse perfectamente en comparación con la gráfica de distribución de los datos anteriormente mostrada en apartados previos.

Los clusters creados con KMeans y GaussianMixture parecen mostrar unos resultados considerablemente buenos y similares a lo que se esperaba, con la diferencia de que hay unos puntos a la izquierda que varían en su asignación de un método a otro (siendo más correcto a simple vista el GaussianMixture). Los centroides también son parecidos entre ambas técnicas.

Con todo, la clase setosa parece clasificarse correctamente en todos los métodos, y son las clases virginica y versicolor las que más cambian en su asignación de un algoritmo a otro.

Coeficiente de Silhouette

Esta métrica evalúa qué tan bien separados están los clusters, y cuán compactos son. A pesar de que toma valores entre 0 y 1, las soluciones con un clustering mayor de 0.5 se consideran ya lo suficientemente buenas.

Los valores obtenidos al calcular el índice de Silhouette para los resultados obtenidos con las tres técnicas son los siguientes:

```
--- SILHOUETTE SCORE ---

Silhouette Score KMeans: 0.566
Silhouette Score DBSCAN: 0.572
Silhouette Score GMM: 0.527
```

Por lo tanto, el mejor algoritmo en cuanto a esta métrica es el DBSCAN, seguido de cerca por KMeans. Con un valor un tanto más bajo encontramos el Gaussian Mixture. A pesar de ello, todos los valores dan resultados relativamente altos que coinciden con un buen clustering (ya que son valores mayores a 0.5). Aún así, ningún modelo alcanza un valor excepcionalmente alto (> 0.7), lo que podría deberse a la superposición natural de las clases versicolor y virginica.

Cabe destacar que el hecho de que el índice sea tan alto en DBSCAN, a pesar de la conclusión sacada en el apartado anterior, se debe muy probablemente a que la métrica no tiene en cuenta el ruido; por lo que valora los puntos clasificados en los clusters (que sí están bien separados y son compactos, al no asignar a ningún cluster los valores más alejados).

Índice de Calinski-Harabasz

El índice de Calinski-Harabasz mide la dispersión entre y dentro de los clusters. Los valores parten de 0 y no tienen límite superior; un coeficiente alto indica un buen clustering.

```
--- CALINSKI-HARABASZ SCORE ---

Calinski-Harabasz KMeans: 544.840
Calinski-Harabasz DBSCAN: 590.948
Calinski-Harabasz GMM: 473.240
```

De nuevo, DBSCAN vuelve a obtener el mejor desempeño, seguido otra vez por KMeans; indicando clusters densos y bien separados entre sí. Por otra parte, GaussianMixture presenta el valor más bajo, sugiriendo clusters más dispersos y menos compactos. Esto se corresponde con los gráficos mostrados sobre los resultados de cada método de clustering (siendo DBSCAN el que tiene los clusters mejor diferenciados y más compactos, y GaussianMixture el que tiene datos más dispersos y sobrepuestos entre clusters).

Adjusted Rand Score

A diferencia de las métricas anteriores que eran parte de un análisis interno del modelo (que únicamente evaluaban la cohesión y dispersión de los clusters), el Adjusted Rand Score o ARI es una métrica de análisis externo que mide la similitud entre las asignaciones a los clusters por parte de los algoritmos y

las etiquetas reales (que contiene el iris dataset). A diferencia del Rand Index tradicional, el Adjusted Rand Index ajusta el resultado para el azar, lo que evita que modelos aleatorios obtengan puntuaciones altas por casualidad. Toma valores entre 0 y 1; y se considera una buena solución a partir de 0.5.

```
--- ADJUSTED RAND SCORE ---

Adjusted Rand Index KMeans: 0.744
Adjusted Rand Index DBSCAN: 0.556
Adjusted Rand Index GMM: 0.941
```

En esta métrica, todos los algoritmos han tenido resultados relativamente buenos (mayores a 0.5); pero es el GaussianMixture el algoritmo que destaca entre los demás por su excelente rendimiento. Como se había anticipado al analizar las gráficas resultado, el GMM es el algoritmo cuyas asignaciones se asemejan más a las etiquetas esperadas; capturando casi a la perfección la estructura de las clases del dataset iris.

Con un muy buen resultado encontramos el KMeans. Se comentó en el análisis de las gráficas que el resultado era similar al de GMM, con la diferencia de algunos puntos en los que este último presentaba un mejor desempeño en comparación con las clases originales; y esto mismo representa el resultado de 0.744. Se trata de un valor excepcionalmente alto, pero no tan bueno como el de GaussianMixture.

El valor más bajo en este caso es el de DBSCAN. Esto sugiere que, a pesar de tratarse de un valor relativamente bueno por ser mayor a 0.5, el hecho de detectar ruido le ha penalizado en gran medida en la evaluación de esta métrica.

Conclusiones

En definitiva, todos los métodos examinados han dado en mayor o menor medida buenos resultados en las métricas evaluadas para determinar el desempeño del clustering.

El **Gaussian Mixtures** fue el mejor modelo en recuperar las clases reales de flores iris, con el ARI más alto y la mejor ejecución al observar las gráficas de matplotlib. El hecho de que sea el mejor método en las métricas de evaluación externas sugiere que el modelo capta bien la estructura subyacente de los datos, aunque los clusters puedan estar más solapados (y por ello tenga los peores resultados en el coeficiente de Silhouette y el índice de Calinski-Harabasz).

Por otro lado, el **DBSCAN** obtiene los índices más altos en métricas de evaluación internas; dado que no evalúa el ruido al medir la cohesión y dispersión entre los clusters y tiende a dar mejores resultados en estas métricas. Difiere en gran medida de sus métricas de evaluación externas, en las que es el peor de los algoritmos. Esto sugiere que se están generando agrupaciones diferentes a las clases reales.

El **KMeans**, sin embargo, tiene un equilibrio entre todas las métricas, lo que lo convierte en una opción sólida para esta solución. Los resultados son lo suficientemente óptimos en las evaluaciones, tanto internas como externas, hechas sobre el dataset.

Por lo tanto, si se desean recuperar las clases reales, GaussianMixtures es el mejor algoritmo; si se busca que los clusters estén bien definidos y separados, DBSCAN es la mejor opción; y si lo que se necesita es un balance entre ambas, KMeans es la opción apropiada.

Bibliografía

Scikit-learn developers. (n.d.). Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/

Hunter, J. D., & Matplotlib Development Team. (n.d.). *Matplotlib: Visualization with Python*. https://matplotlib.org/stable/index.html

NumPy Developers. (n.d.). *NumPy documentation*. https://numpy.org/doc/

Link al código utilizado

https://github.com/saraportto/APAU_II/blob/main/P2_Clustering_Methods/P2_clustering_methods.py