Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 03/26/2022
Practica No.6 Clustering Jerárquico y Particional

Objetivo.

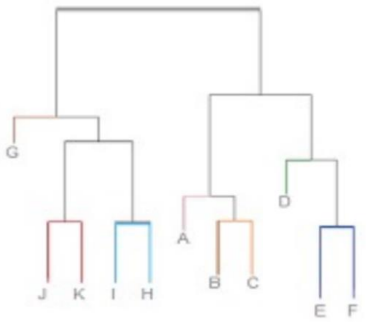
Obtener grupos de pacientes con características similares, diagnosticadas con un tumor de mama, a través de clustering jerárquico y particional.

Características.

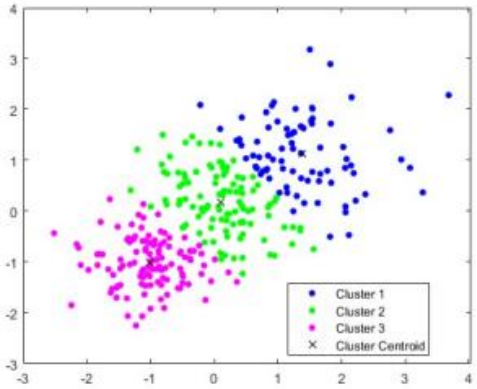
Estudios clínicos a partir de imágenes digitalizadas de pacientes con cáncer de mama de Wisconsin (WDBC, Wisconsin Diagnostic Breast Cancer).

El clustering lo que busca es hacer una segmentación de los registros en el conjunto de datos, con esto se descubren una serie de patrones que normalmente estarían ocultos los cuales relacionan a cada uno de ellos consiguiendo agruparlos, sin embargo, no se ofrece una descripción clara y es tarea de nosotros darles una interpretación adecuada.

En particular el clustering jerárquico organiza los datos en una estructura de tipo árbol de forma recursiva de tal manera que en cada nivel del árbol se tiene cierto número de clústeres no definido e incrementa conforme bajamos en el árbol.



Por otro lado, el clustering particional organiza a los elementos determinando n centroides (Punto que ocupa la posición media en un clúster), donde n es un número que tenemos que definir nosotros, se usara el algoritmo de k-means el cual va a estar moviendo iterativamente los centroides tomando la media de cada clúster que se forme hasta un punto de convergencia donde ya no se muevan más.



Un punto importante de este algoritmo es que tenemos que definir el número de clústeres con los cuales queremos terminar y para ello es necesario utilizar otras herramientas como el método del codo, de la rodilla o el clustering jerárquico de la practica anterior los cuales nos sugieren un buen número por el cual podríamos iniciar con nuestro análisis.

Desarrollo.

Como primer paso tenemos la importación de las librerías necesarias para trabajar con el conjunto de datos de los pacientes a segmentar. `pandas` para la manipulación de los datos, `numpy` para el manejo de matrices, `matplotlib` y `seaborn` para la visualización de estos datos mediante gráficos de diferente tipo dependiendo del análisis, `StandardScaler` y `MinMaxScaler` para escalar o normalizar el conjunto de datos evitando sesgos en el análisis y aumentando también el rendimiento de los algoritmos.

Para el algoritmo de clustering jerárquico necesitaremos de `scipy` a la librería `hierarchy` para la implementación del modelo para el clustering jerárquico y además `AgglomerativeClustering` para etiquetar a cada registro con su respectivo cluster.

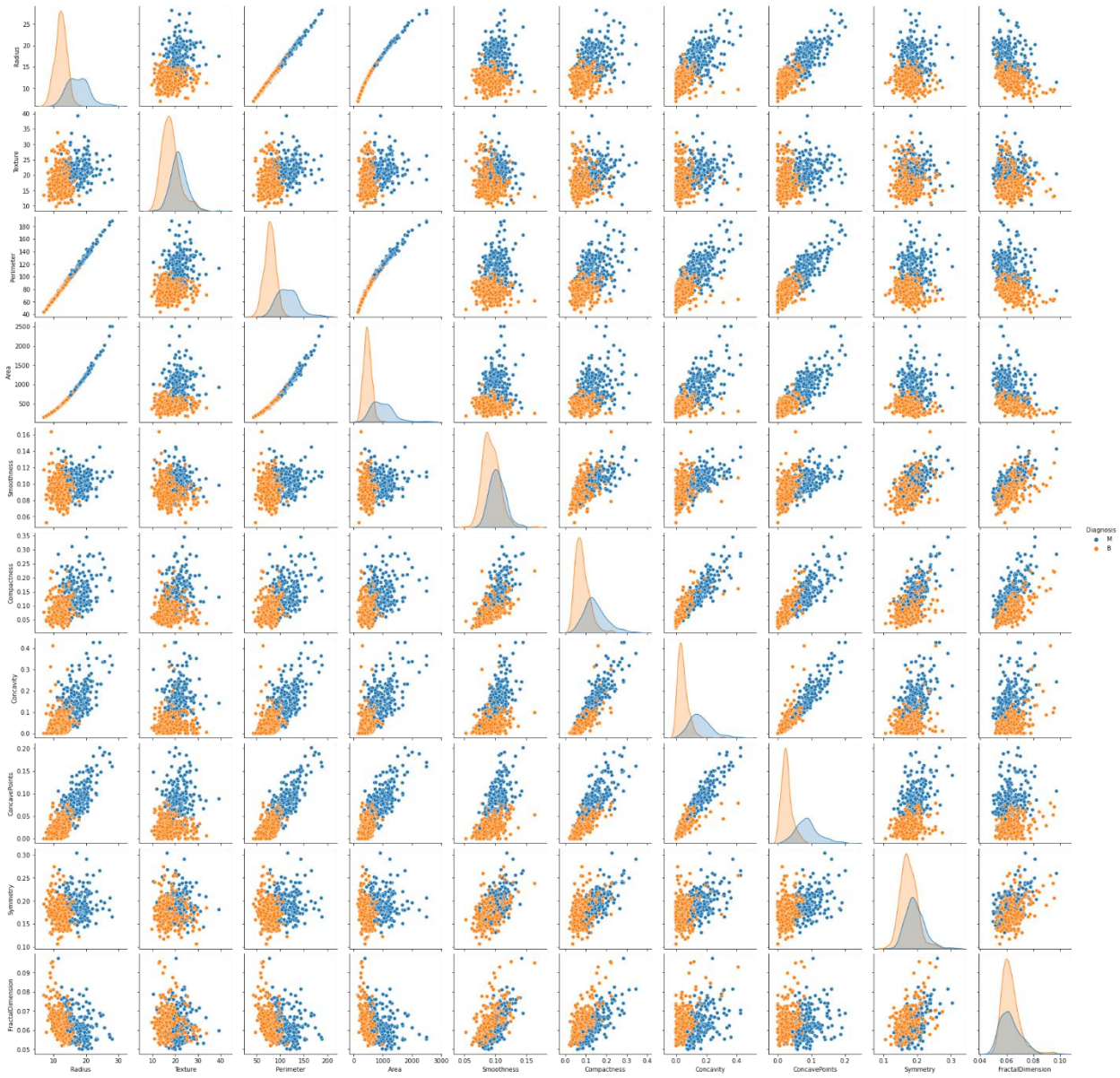
Por la parte del clustering particional necesitaremos además instalar la librería `kneed` para el método de la rodilla el cual nos da una aproximación adecuada de cual deberá de ser el número de clústeres para el algoritmo de k-means y por último la librería de `KMeans` que se encuentra dentro de los paquetes que ofrece `sklearn` para estos desarrollos.

Nuestro `DataFrame` leído con ayuda de `pandas` se ve de la siguiente forma con 12 columnas y 569 registros.

	IDNumber	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809
...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

Como podemos apreciar la mayoría de los datos son numéricos a excepción de `IDNumber` y `Diagnosis` los cuales son objetos, por esta misma razón no se considerarán para el análisis de correlación que se tiene que hacer para identificar variables que pudieran estar aportando la misma información al modelo.

Para detectar este tipo de relación entre variables lo primero que se hizo fue trazar el grafico de dispersión de cada una de las características del modelo para observar a simple vista si es que hay candidatos a ser eliminados, este grafico lo podemos ver claramente en la siguiente página.



En la matriz podemos ver que claramente hay 2 conjuntos de variables los cuales parecen estar fuertemente relacionados; `Radius` con `Perimeter` y `Concabity` con `CancavePoints`. Para asegurarnos de estas suposiciones se calculó la matriz de correlaciones con el método `corr` del `DataFrame`, con esta matriz nosotros podemos ordenar las columnas para que muestre la relación que existe para determinada variable con ayuda del método `sort_values` pasando la columna del `DataFrame` que queramos ordenar o bien podemos pasar esta matriz de correlaciones como parámetro a un mapa de calor el cual pinta de colores claros o fríos los índices de interés, esto se implementa con `seaborn` para observar más claro cuáles son los índices de correlación altos ya que al ser una matriz numérica de 10x10 es un poco complicado identificarlos y se tiende a cometer errores.

Al final la selección de características para el modelo quedo de la siguiente manera.

- Radius [Se elimino por su relación con área]
- **Texture [Se conserva]**
- Perimeter [Se elimino por su relación con área]
- **Area [Se conserva]**
- **Smoothness [Se conserva]**
- **Compactness [Se conserva]**
- Concavity [Se elimina por tener una relación fuerte con Concavity, Perimeter, Area, etc.]
- ConcavePoints [Se elimina por tener una relación fuerte con Concavity, Perimeter, Area, etc.]
- **Simmetry [Se conserva]**
- **FractalDimension [Se conserva]**

Con la selección de características completa lo que restaría seria hacer un nuevo `DataFrame` con esas columnas (569x6) para después hacer el proceso de estandarizar o normalizar los datos con el objetivo de que cada uno de éstos aporte la misma información a los modelos evitando sesgos y aumentando el rendimiento que se pudiera tener. Estos procesos se hacen con instancias de las clases `StandardScaler` (Estandarizar) y `MinMaxScaler` (Normalizar). A continuación, se muestran las matrices resultantes.

Estandarización. (Media = 0, Desviación Estándar = 1 y utilizando `StandardScaler.fit_transform`).

	0	1	2	3	4	5
0	-2.073335	0.984375	1.568466	3.283515	2.217515	2.255747
1	-0.353632	1.908708	-0.826962	-0.487072	0.001392	-0.868652
2	0.456187	1.558884	0.942210	1.052926	0.939685	-0.398008
3	0.253732	-0.764464	3.283553	3.402909	2.867383	4.910919
4	-1.151816	1.826229	0.280372	0.539340	-0.009560	-0.562450
...	...	...	...	...	...	...
564	0.721473	2.343856	1.041842	0.219060	-0.312589	-0.931027
565	2.085134	1.723842	0.102458	-0.017833	-0.217664	-1.058611
566	2.045574	0.577953	-0.840484	-0.038680	-0.809117	-0.895587
567	2.336457	1.735218	1.525767	3.272144	2.137194	1.043695
568	1.221792	-1.347789	-3.112085	-1.150752	-0.820070	-0.561032

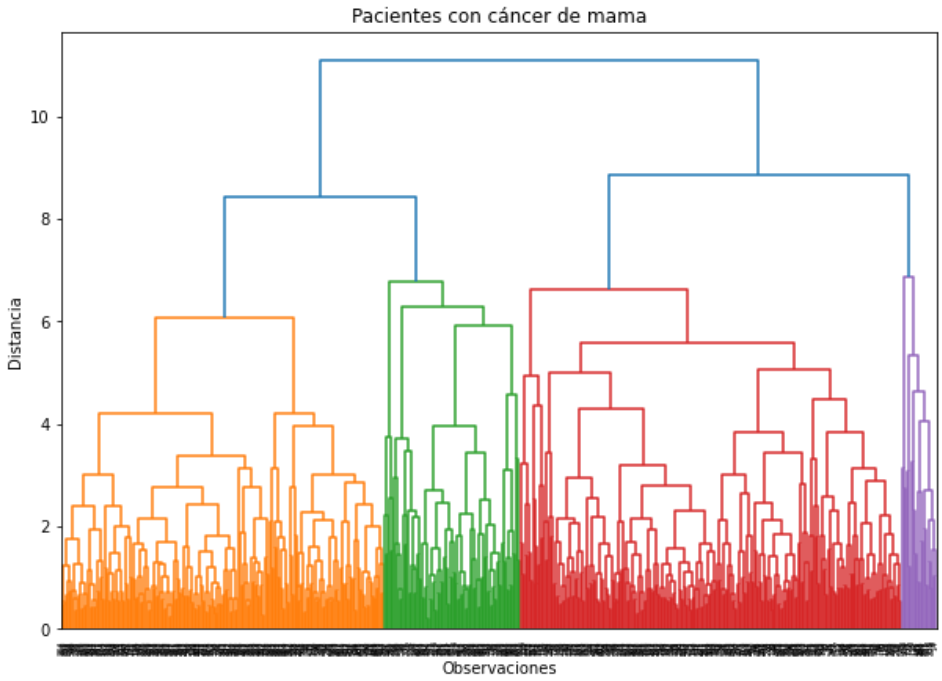
Normalización. (Escala de valores entre 0 y 1 utilizando `MinMaxScaler.fit_transform`).

	0	1	2	3	4	5
0	0.022658	0.363733	0.593753	0.792037	0.686364	0.605518
1	0.272574	0.501591	0.289880	0.181768	0.379798	0.141323
2	0.390260	0.449417	0.514309	0.431017	0.509596	0.211247
3	0.360839	0.102906	0.811321	0.811361	0.776263	1.000000
4	0.156578	0.489290	0.430351	0.347893	0.378283	0.186816
...	...	...	...	...	...	...
564	0.428813	0.566490	0.526948	0.296055	0.336364	0.132056
565	0.626987	0.474019	0.407782	0.257714	0.349495	0.113100
566	0.621238	0.303118	0.288165	0.254340	0.267677	0.137321
567	0.663510	0.475716	0.588336	0.790197	0.675253	0.425442
568	0.501522	0.015907	0.000000	0.074351	0.266162	0.187026

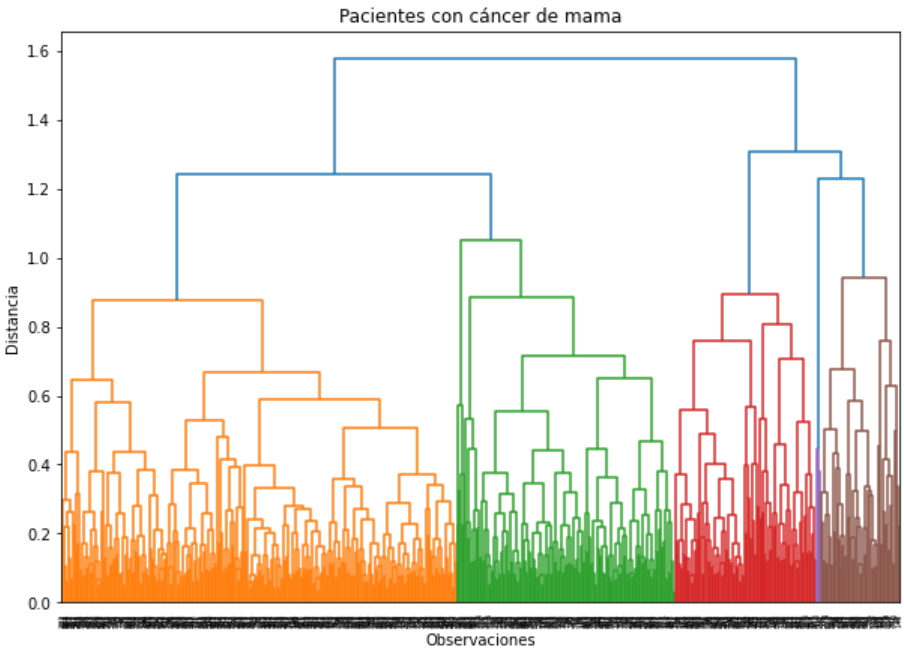
Con nuestros datos ya en forma podemos comenzar con la aplicación de los algoritmos iniciando con el clustering Jerárquico Particional el cual solamente necesita como entrada la matriz de datos con los registros que se van a segmentar y la métrica de distancias que queremos usar ayudándonos de la librería de `scipy` en su modulo `hierarchy` para trazar el `dendrograma` o grafico de árbol de la segmentación el cual se generó visualmente con ayuda de `matplotlib`. A continuación, se muestran los resultados de los clusters obtenidos para las distancias euclidiana, chebyshev y Manhattan.

```
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
plt.figure(figsize=(10, 7))
plt.title("Pacientes con cáncer de mama")
plt.xlabel('Observaciones')
plt.ylabel('Distancia')
Arbol = shc.dendrogram(shc.linkage(MNormalizada, method='complete', metric='cityblock'))
```

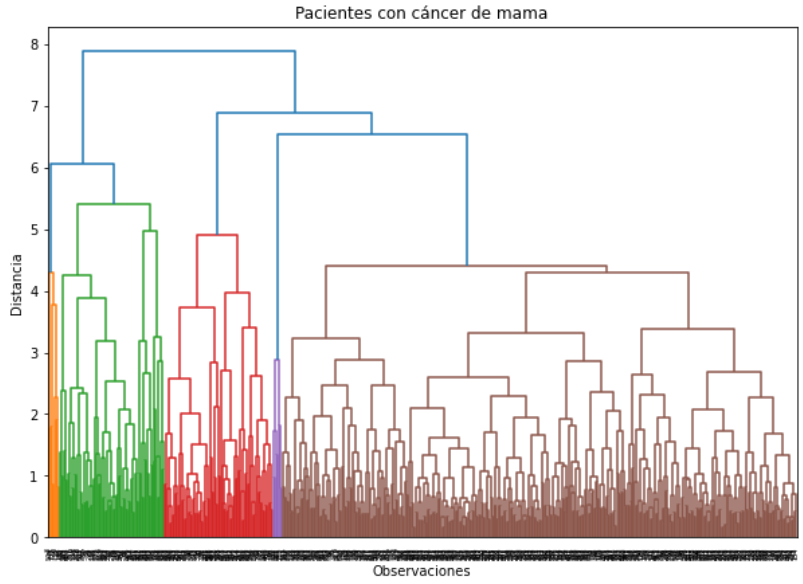
Métrica de distancia: `euclidean`, Matriz: `Estandarizada`, Resultado: `4 clústeres`.



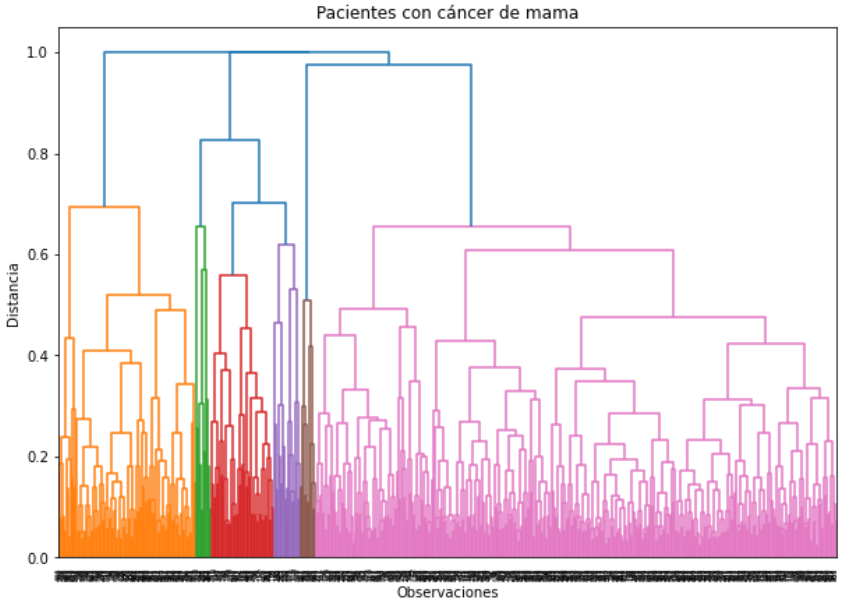
Métrica de distancia: `euclidean`, Matriz: `Normalizada`, Resultado: `5 clústeres`.



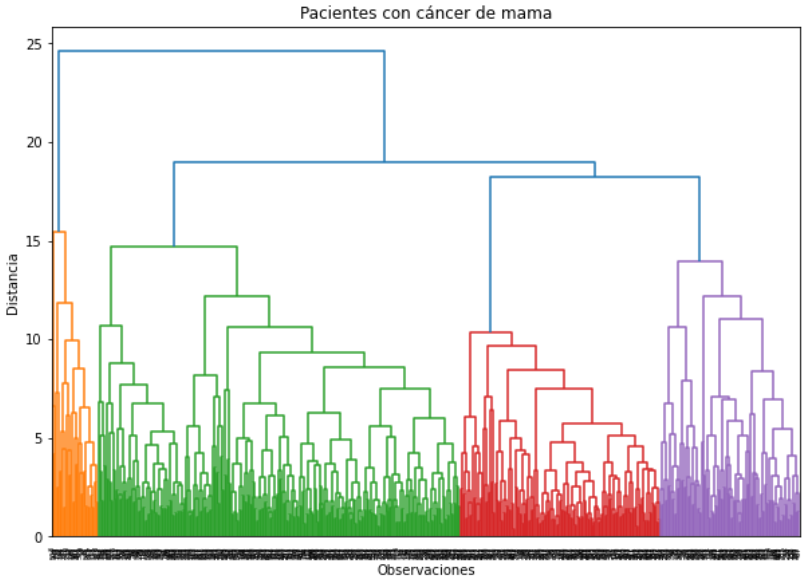
Métrica de distancia: `chebyshev`, Matriz: `Estandarizada`, Resultado: `5 clústeres`.



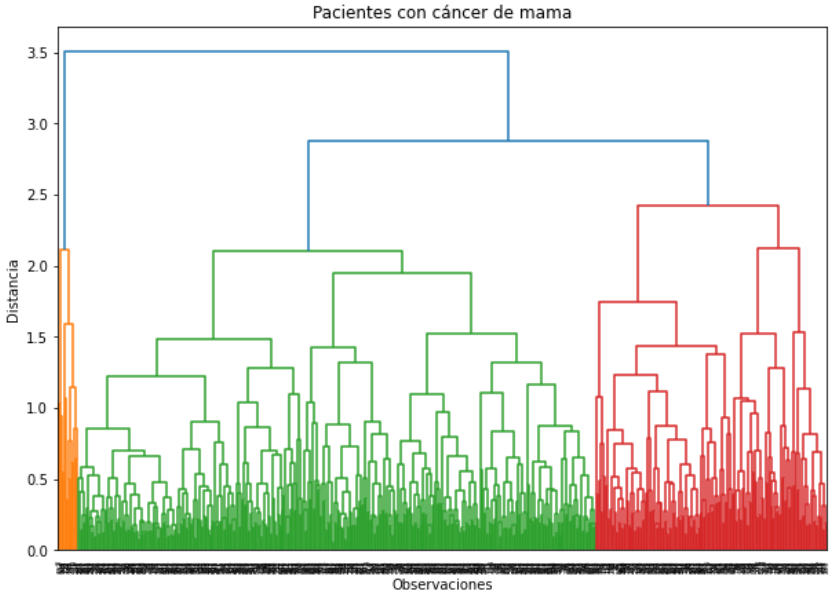
Métrica de distancia: `chebyshev`, Matriz: `Normalizada`, Resultado: `6 clústeres`.



Métrica de distancia: `cityblock`, Matriz: `Estandarizada`, Resultado: `4 clústeres`.



Métrica de distancia: `cityblock`, Matriz: `Normalizada`, Resultado: `3 clústeres`.



Con nuestros estos árboles podemos etiquetar a cada uno de los registros en el conjunto de datos ayudándonos de la clase `AgglomerativeClustering` y pasando como parámetros el número de clústeres, el método y la distancia que pasamos al hacer los dendrogramas, con esto se crea un objeto el cual lo podemos usar para la segmentación con su método `fit_predict` y pasando como parámetro la matriz de datos con la que se trabajara. Para esta parte del análisis se considerarán los clústeres obtenidos con la matriz estandarizada y la métrica euclidiana la cual nos da como resultado 4 clústeres. El primero de ellos con 23 registros, el segundo con 88, el tercero con 248 y el cuarto con 120 registros.

clusterH	
0	23
1	88
2	248
3	210

```
MJerarquico = AgglomerativeClustering(n_clusters=4, linkage='complete', affinity='euclidean')
MJerarquico.fit_predict(MEstandarizada)
MJerarquico.labels_
```

Con cada uno de los registros etiquetados se agrupa por clúster y luego se obtienen los centroides calculando la media para cada uno de los registros los cuales se muestran a continuación.

clusterH	Texture	Area	Smoothness	Compactness	Symmetry	FractalDimension
0	20.133478	775.543478	0.124274	0.242200	0.240830	0.077839
1	22.540568	1243.728409	0.098441	0.137140	0.182560	0.058889
2	18.167540	561.336694	0.103316	0.114235	0.190486	0.065737
3	19.160095	505.403810	0.084217	0.063813	0.163030	0.059317

Para cada uno de estos clústeres es pertinente hacer un análisis que describa de forma textual como es el comportamiento o las características que distinguen a cada uno de estos grupos.

Clúster 0.

Conformado por 23 pacientes con indicios de cáncer maligno por el tamaño del tumor, con un área promedio de tumor de 775 pixeles y una desviación estándar de textura de 20 pixeles. Aparentemente es un tumor compacto (0.24 pixeles), cuya suavidad alcanza 0.12 pixeles, una simetría de 0.24 y una aproximación de frontera, dimensión fractal, promedio de 0.077 pixeles.

Clúster 1.

Conformado por 88 pacientes con muy altas probabilidades de cáncer maligno por el tamaño del tumor, con un área promedio de tumor de 1244 pixeles y una desviación estándar de textura de 23 pixeles. Aparentemente es un tumor compacto (0.13 pixeles), cuya suavidad alcanza 0.09 pixeles, una simetría de 0.18 y una aproximación de frontera, dimensión fractal, promedio de 0.058 pixeles.

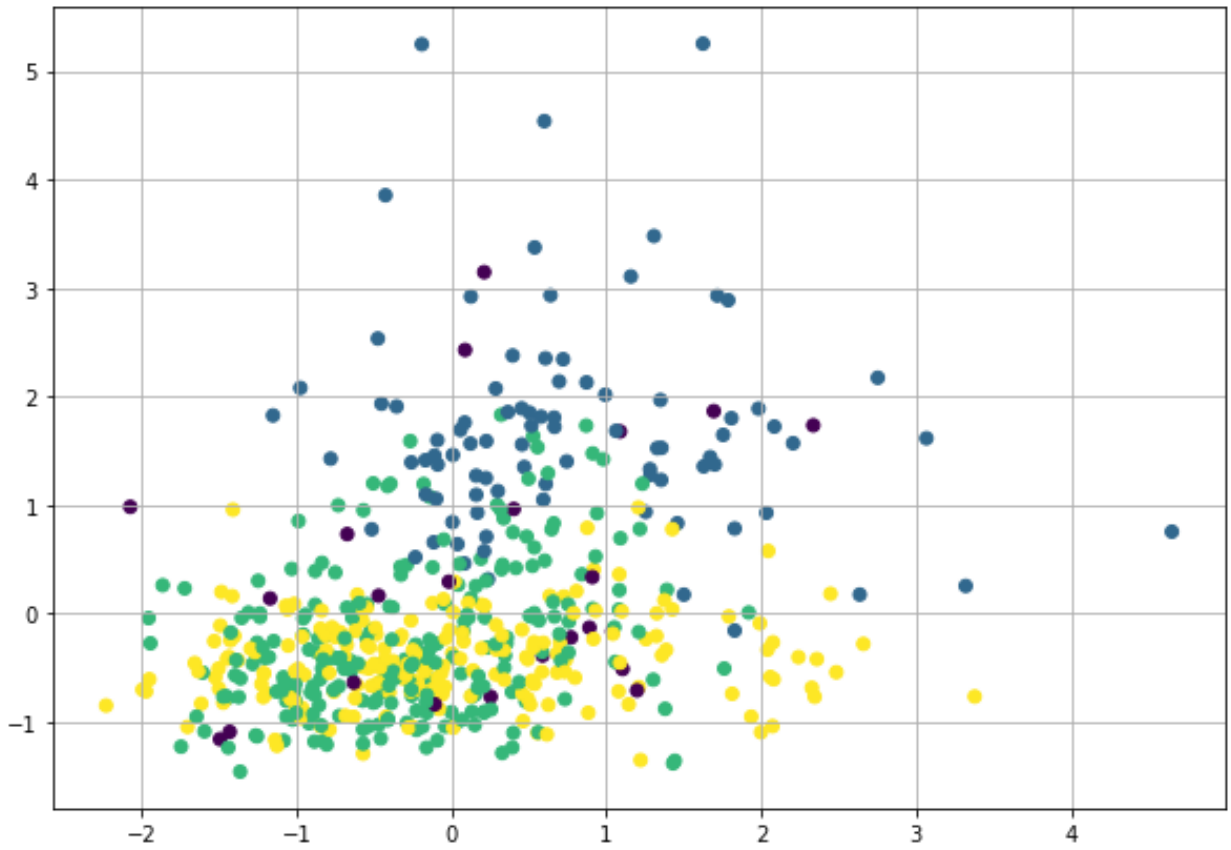
Clúster 2.

Conformado por 248 pacientes con bajas probabilidades de cáncer maligno por el tamaño del tumor, con un área promedio de tumor de 561 pixeles y una desviación estándar de textura de 18 pixeles. Aparentemente es un tumor compacto (0.11 pixeles), cuya suavidad alcanza 0.1 pixeles, una simetría de 0.19 y una aproximación de frontera, dimensión fractal, promedio de 0.065 pixeles.

Clúster 3.

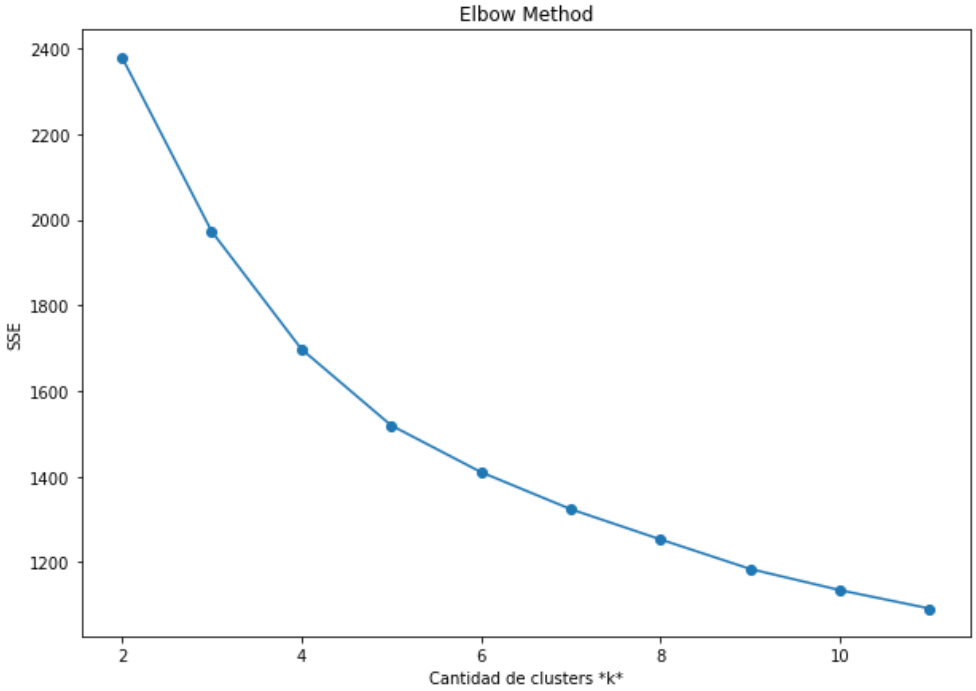
Es un grupo formado por 210 pacientes con el menor tamaño de tumor (posiblemente benigno), con un área promedio de tumor de 505 pixeles y una desviación estándar de textura de 19 pixeles. Es un tumor compacto (0.06 pixeles), cuya suavidad alcanza 0.08 pixeles, una simetría de 0.16 y una aproximación de frontera, dimensión fractal, promedio de 0.059 pixeles.

Podemos además trazar un gráfico de dispersión para darnos una idea de como es que se distribuyen estos clústeres con ayuda de `matplotlib` y la matriz de datos estandarizada ya etiquetada para cada registro.

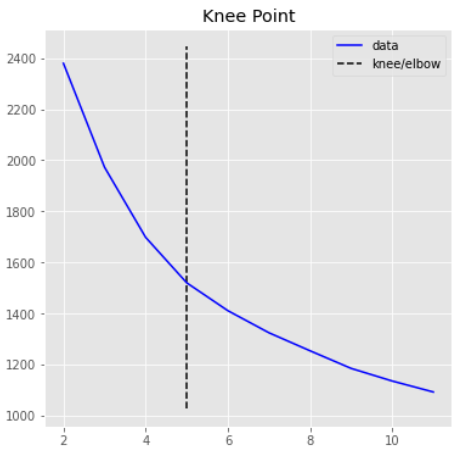


Comenzando con el clustering particional primero tenemos que establecer cual es el numero óptimo de clústeres a utilizar, podemos intuir por los arboles generados en el clustering jerárquico que este número pudiera estar entre 4 y 5 clústeres.

Para establecer bien este numero es necesario ayudarnos en primera instancia del método del codo calculando la suma de las distancias al cuadrado desde cada uno de los registros a su centroide para cada una de las configuraciones, afortunadamente la clase de `KMeans` que nos proporciona la biblioteca de `sklearn` ya calcula estas sumas para la configuración de clústeres que se le da y basta con aplicar el modelo a la matriz de datos con el método `fit` para que estas sumas se dejen en el atributo `inertia_` el cual se ingresa a una lista ya que se hace para las configuraciones de 2 a 12 clústeres que es lo recomendado. Por ultimo se grafican estas sumas con ayuda de `matplotlib`.



Como podemos apreciar en esta ocasión el cambio abrupto de dirección que indicaría el número de clústeres adecuado no es tan claro pudiendo estar en 4, 5 o hasta 6 clústeres. En este caso el método de la rodilla del paquete de `Kneed` nos servirá para tener mayor seguridad de que el número de clústeres que se elija sea el adecuado. Basta con hacer un objeto de la clase `KneeLocator` el cual recibirá como parámetros el rango de clústeres, la lista de distancias para cada uno de esos clústeres, la forma de la curva (`convex`) y la dirección que tomara (`decreasing`) obteniendo que el numero adecuado de clústeres es de 5 y la grafica que se muestra a la derecha marcando este punto con una linea.



Ya con el número de clústeres definido, el siguiente paso seria aplicar el algoritmo con la clase de `KMeans` de `sklearn` para esta configuración, primero creando un objeto y luego calculándolo con el método `predict` para poder tener el atributo l`abels_` disponible el cual contiene una lista ordenada con las etiquetas de todos los clústeres del modelo, en esta ocasión como son 5 clústeres entonces se numerarán del 0 al 4. Esta es una lista que fácilmente se puede agregar como una columna mas al `DataFrame` de la matriz de datos resultante de la reducción de características del modelo

```
MParticional = KMeans(n_clusters=5, random_state=0).fit(MEstandarizada)
MParticional.predict(MEstandarizada)
MParticional.labels_
```

clusterP	
0	85
1	48
2	184
3	153
4	99

Con los registros etiquetados podemos agruparlos y contarlos con ayuda de pandas, de esta forma obtenemos que en el primer clúster se tienen 85 elementos, en el segundo 48, en el tercero 184, en el cuarto 153 y en el ultimo 99 registros de pacientes.

Podemos además con ayuda de esta agrupación calcular los promedios por cada grupo para obtener a cada uno de sus centroides correspondientes obteniendo los siguientes resultados.

clusterP	Texture	Area	Smoothness	Compactness	Symmetry	FractalDimension
0	24.492706	559.569412	0.085045	0.074626	0.164491	0.059430
1	20.746875	738.941667	0.117829	0.211744	0.229617	0.075797
2	16.339891	511.619022	0.086801	0.063990	0.165405	0.059257
3	17.822680	480.716993	0.105121	0.112265	0.190299	0.067212
4	21.865354	1231.431313	0.099894	0.140528	0.187147	0.059145

Clúster 0.

Es un grupo formado por 85 pacientes con un tamaño pequeño de tumor (potencialmente benigno), con un área promedio de tumor de 559 pixeles y una desviación estándar de textura de 24 pixeles. Es un tumor compacto (0.07 pixeles), cuya suavidad alcanza 0.08 pixeles, una simetría de 0.16 y una aproximación de frontera, dimensión fractal, promedio de 0.059 pixeles.

Clúster 1.

Es un grupo formado por 48 pacientes con tamaño de tumor grande (potencialmente maligno), con un área promedio de tumor de 739 pixeles y una desviación estándar de textura de 21 pixeles. Es un tumor no tan compacto (0.21 pixeles), cuya suavidad alcanza 0.12 pixeles, una simetría de 0.22 y una aproximación de frontera, dimensión fractal, promedio de 0.075 pixeles.

Clúster 2.

Conformado por 184 pacientes con bajas probabilidades de cáncer maligno por el tamaño del tumor, con un área promedio de tumor de 512 pixeles y una desviación estándar de textura de 16 pixeles. Aparentemente es un tumor compacto (0.06 pixeles), cuya suavidad alcanza 0.09 pixeles, una simetría de 0.16 y una aproximación de frontera, dimensión fractal, promedio de 0.059 pixeles.

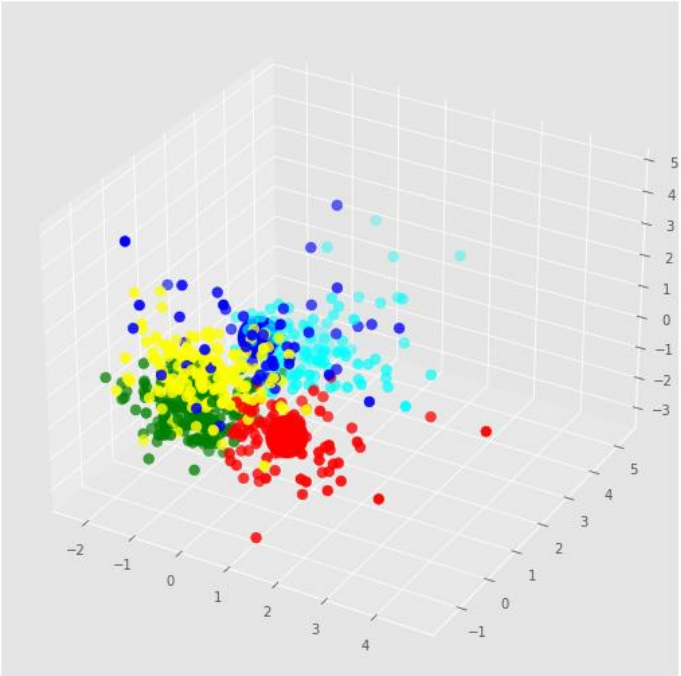
Clúster 3.

Es un grupo formado por 153 pacientes con bajas probabilidades de cáncer maligno por el tamaño del tumor, con el área promedio de tumor más baja de 480 pixeles y una desviación estándar de textura de 18 pixeles. Aparentemente es un tumor compacto (0.11 pixeles), cuya suavidad alcanza 0.11 pixeles, una simetría de 0.19 y una aproximación de frontera, dimensión fractal, promedio de 0.067 pixeles.

Clúster 4.

Conformado por 99 pacientes con muy altas probabilidades de cáncer maligno por el tamaño del tumor, ya que tiene un área promedio de tumor de 1231 pixeles y una desviación estándar de textura de 22 pixeles. Aparentemente es un tumor compacto (0.14 pixeles), cuya suavidad alcanza 0.099 pixeles, una simetría de 0.19 y una aproximación de frontera, dimensión fractal, promedio de 0.059 pixeles.

El último paso del análisis es el generar una gráfica en la cual podemos ver un poco de la distribución de nuestros 5 clústeres de datos la cual se crea partiendo de la matriz de datos estandarizada y se ve de la siguiente forma.



Conclusiones.

La técnica de clustering particional puede llegar a resultar muy útil cuando lo que se busca es una segmentación con un mayor número de registros ya que cuando este número incrementa la implementación de un árbol para organizar a los clústeres (Clustering Jerárquico) se vuelve inestable y más lenta donde el hecho de estandarizar y normalizar los datos si bien puede ayudar al rendimiento no lo hace a tal punto de hacer viable el clustering jerárquico, por el contrario hay que tener en cuenta que para poder utilizar la técnica particional se tiene que conocer cuál es el número de clústeres con el que queremos terminar, métrica que es muy importante ya que si se aplica mal podría llevar a tener sesgos en el análisis de los clústeres pero que requiere de un procesamiento extra para realizarse (Método del codo o la rodilla).

Si el numero de datos nos lo permite pudiéramos también implementar el clustering Jerárquico para hacer la aproximación del número de clústeres adecuado para el algoritmo a usar en clustering particional pues si observamos los árboles obtenidos a partir del clustering particional con la matriz estandarizada para cada una de las distancias obtendremos que el número de clústeres adecuado es de 4 y 5 clústeres (dependiendo la métrica) y al realizar el método de la rodilla para estos mismos datos obtuvimos que el numero indicado de igual manera es de 4 clústeres.

Fuentes.

Datos: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

sklearn.cluster.KMeans. (2022). Scikit-Learn. 2022, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html