Proyecto 3

1st Ian Augusto Cortez Gorbalan
ian.cortez@utec.edu.pe
970895554

2nd Plinio Matías Avendaño Vargas
plinio.avendano@utec.edu.pe
927144823

I. Introducción

Este proyecto busca emplear los siguientes algoritmos no supervisados de clustering, *K-Means*, DBSCAN, GMM y Agglomerative Hierarchical Clustering con el objetivo de agrupar los datos de un *dataset* real. En el presente trabajo se va a emplear un *dataset* de información genética con el propósito de congregar los tejidos similares en grupos lo mejor posible. Esta data cuenta con 189 elementos cada uno con 22215 características, además de los nombres para cada una de las muestras de los tejidos y lo que representan las características de cada elemento. El objetivo principal es agrupar de forma correcta los tejidos que sean similares en base a las características de cada elemento del *dataset*. Como objetivos secundarios, se busca implementar correctamente los algoritmos de *clustering* mencionados. Adicionalmente, se busca determinar cuál de los algoritmos de *clustering* mencionados es capaz de agrupar de mejor forma los elementos del *dataset*, esto se hara mediante el calculo del **Rand Index** de cada algoritmo, de esta forma se podran comparar objetivamente.

II. Explicación de los modelos empleados

A. K-Means

El algoritmo de *K-Means* es un algoritmo de *clustering* no supervisado que agrupa elementos de un *dataset* en base a la distancia de estos elementos a un centroide para generar k *clusters* con los centroides.

El primer paso para agrupar los datos es definir la cantidad k de centroides y *clusters*. Posterior a definir la cantidad, se toman k datos aleatorios del *dataset* como los centroides iniciales y se asignan los puntos más cercanos a cada uno de los centroides para formar los *clusters*.

El segundo paso es calcular el promedio de todos los valores que pertenecen a cada uno de los *clusters* con:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q$$

Donde μ_k representa el promedio del *cluster* k, N_k representa la cantidad de elementos que pertenecen a ese cluster y x_q representa un punto dentro del *cluster*.

Después de calcular el promedio, se asignan los elementos del *dataset* más cercanos a cada uno de los centroides. Para esto se calcula la distancia entre cada uno de los centroides y un elemento del dataset y se le asigna el punto al centroide más cercano. Al terminar, se actualizan los centroides basados en el promedio de los valores calculados para cada uno de los elementos del centroide. Se repite el segundo paso hasta que no haya ningun cambio en el valor de los centroides o hasta pasar un *threshold*.

B. GMM

El algoritmo de Gaussian Mixture Model, como su nombre lo indica consiste en una superposición lineal de K campanas de Gauss, en el cual se busca maximizar la probabilidad conjunta de que cada uno de los samples pertenezcan a alguna de las campanas, esto se consigue mediante la modificación de la media y la matriz de covarianza de cada una de las distribuciones normales, es decir estamos buscando la forma mas óptima de ajustar un modelo de distribución a la data, lo que también se denomina *Maximum Likelihood*.
Dado este modelo, lo que buscamos maximizar es la productoria de la probabilidad de cada sample de pertenecer a alguna de las K campanas, en términos matemáticos $p(dataset) = \prod_{j=1}^{n} p(x_j)$.
Siendo $p(x_j) = \sum_{i=1}^{K} p(x_j, Z_i = 1)$
Por lo que por teorema de Bayes, se puede descomponer a lo siguiente.
$p(x_j, Z_i = 1) = p(x_j|Z_i)p(Z_i = 1) = \pi_i N(x_j|u_i, \sum_i)$
Siendo pi la probabilidad a priori de la i-th gaussiana.
Expandiendo este modelo resultamos en que buscamos maximizar, $p(dataset) = \prod_{j=1}^{n}(\sum_{i=1}^{K} \pi_i N(x_j|u_i, \sum_i))$
Sin embargo, debido a que es una productoria, tendera a un overflow rápidamente, por tanto aplicando logaritmo logramos reducir el computo de esta operacion.
$ln(p(dataset)) = ln(\prod_{j=1}^{n}(\sum_{i=1}^{K} \pi_i N(x_j|u_i, \sum_i)))$
$ln(p(dataset)) = \sum_{j=1}^{n}(ln(\sum_{i=1}^{K} \pi_i N(x_j|u_i, \sum_i)))$
Lo cual se denomina log likelihood.
Debido a que las variables aleatorias no son derivables, para entrenar este modelo se empleara el algoritmo de Expectation-Maximization, este consiste en introducir una nueva variable denominada y la cual sera una matriz de n*k, en la cual cada elemento es la probabilidad de que el ith sample venga de la j campana o cluster, el paso en el que se genera esta variable es el de expectation y en maximization empleamos la nueva variable para fijar como constante y calcular las nuevas, covarianza, media y π.
Entonces, en expectaction se crea la matriz de tamaño n*k, operando de la siguiente manera, $y(z_n k) = \frac{\pi_k N(x_n|u_k, \sum_k)}{(\sum_{i=1}^{K} \pi_i N(x_n|u_i, \sum_i)}$
Mientras que en maximization, se actualizan los parámetros de las campanas normales, $u_k = \frac{\sum_{i=1}^{N} y(Zik) x_i}{N_k}$, $\sum_k = \frac{\sum_{i=1}^{N} y(Zik)(x_i - u_K)(x_i - u_k)^T}{N_k}$ y $\pi_k = \frac{N_k}{N}$ siendo, $N_K = \sum_{i=1}^{N} y(Z_i k)$

C. DBSCAN

El algoritmo de *DBSCAN* es un algoritmo de *clustering* no supervisado que agrupa elementos de un *dataset* en un *cluster* en base a la distancia entre los elementos dentro de un gráfico K-dimensional y la cantidad de puntos cercanos entre sí.

El algoritmo toma como hiper-parámetros el radio de cada cluster y la cantidad mínima de puntos. Adicionalmente, se toman como *inputs* la base de datos y la función para determinar la distancia entre puntos.

DBSCAN empieza por recorrer todos los puntos del *database*. Primero, para cada punto, revisa si ha sido asignado a un *cluster* en base a un *label* para evitar volver a procesar puntos. Después, encuentra los vecinos para el punto elegido. Si la cantidad de vecinos para un punto es menor a la cantidad mínima de puntos, se marca el punto actual como ruido (elemento que no tiene suficientes vecinos en un radio determinado) y se avanza al siguiente punto. Luego, se genera un nuevo *cluster* con un *label* determinado que se le asigna al punto actual.

A continuación, se genera un *seed* que contiene a todos los vecinos encontrados para el punto actual. Se toma un nuevo punto del grupo y se verifica si tiene un *label* que representa ruido para asignar este punto al *cluster* actual y en el caso de que este definido sin ser ruido, se saltea el elemento. Después de esto, se buscan los vecinos para este punto y se colocan en una lista. Posteriormente, se le da un *label* correspondiente al *cluster* actual, se verifica que la cantidad de vecinos sea el mínimo requerido y se agregan los vecinos encontrados al *seed*, en caso de que no halla la cantidad mínima de vecinos se continua al siguiente elemento del *seed*.

D. Aglomerative Hierarchical Clustering

El algoritmo a explicar, empieza considerando cada sample como un cluster, para después unir los clusters similares (basados en una función de asimilitud computada offline) y combinarlos en uno solo, esto se ejecuta iterativamente hasta que todos los samples pertenezcan a un mismo cluster.

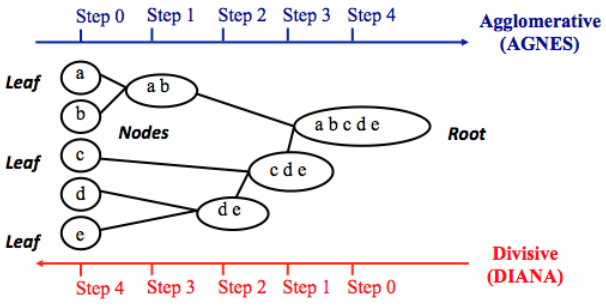

Fig. 1: Representacion de Aglomerative Hierarchical Clustering

E. Principal Component Analysis

El presente algoritmo tiene como principal objetivo disminuir la cantidad de dimensiones de un dataset, calcula la media de cada característica con el objetivo de realizar una matriz de covarianza, para después calcular los eigenvectors y eigenvalues de cada dimensión, esto corresponde a la dirección de mayor varianza en la data (componente principal) y al escalar que indica cuanta varianza existe en ese vector, respectivamente. Una vez obtenido estos vectores, solo queda sortear por sus eigenvalues, para así obtener las características de mayor a menor relevancia.

III. Experimentación

A. Procesamiento de datos

1) Reducción de dimensionalidad: Es vital para el correcto comportamiento del proyecto disminuir el numero de features por sample, ya que al tener más de 22215 características, debido a la "maldición de la dimensionalidad", nuestros datos se vuelven esparsos, es decir pierden significancia estadística, lo que indudablemente causa un mal comportamiento en los algoritmos de clustering.

Debido a esto, empleamos el algoritmo PCA para reducir la dimensionalidad a las caracteristicas necesarias para manterner el 95% de varianza en nuestra data lo que equivale a 109 dimensiones.

Cabe destacar que antes de aplicar este algoritmo es necesario estandarizar la data, ya que sino algunas características podrían tener un sesgo tanto positivo como negativo.

B. K-Means

Para este algoritmo se va a trabajar con 7 *clusters* ya que es la cantidad de *labels* existentes. Cabe destacar que para la implementación la condición para terminar el algoritmo es simplemente llegar un punto donde los centroides de cada *cluster* no cambien más. Para evaluar la calidad de los *clusters*, se va a evaluar que cada dato pertenezca al *cluster* correspondiente (lo que representa que tipo de tejido se asume que es) y comparar con el tipo de tejido que es en realidad. La denominación para cada tipo de tejido es el siguiente:

Tejido	Denominación del tipo de tejido
kidney	0
hippocampus	1
cerebellum	2
colon	3
liver	4
endometrium	5
placenta	6

Para esto, se usará la métrica ***Rand Index***. A continuación se muestran 3 experimentos realizados y sus correspondientes valores de ***Rand Index***:

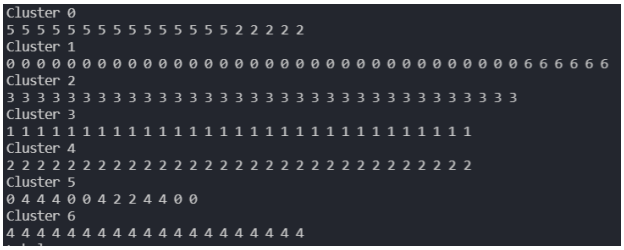

Fig. 2: Clusters para la prueba 1

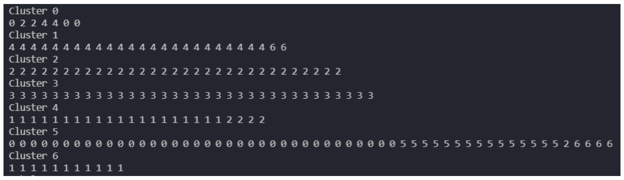

Fig. 3: Clusters para la prueba 2

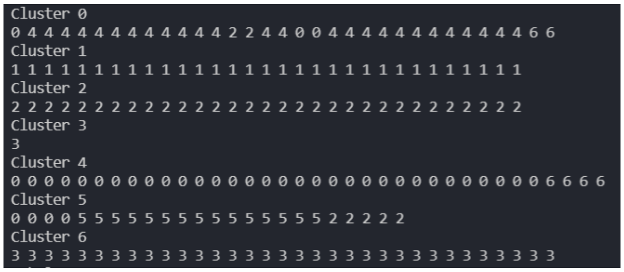

Fig. 4: Clusters para la prueba 3

Prueba	***Rand Index***
1	0.8214159835547213
2	0.6941375904072494
3	0.7972406624451223
Promedio	0.770931412135698

Con los valores del ***Rand Index*** se puede evidenciar que los *clusters* que se han formado por el algoritmo contienen los datos que corresponden al *cluster* correcto. Esto se puede deber principalmente a que el algoritmo se ejecuta hasta que encuentra un centroide óptimo por lo que puede agrupar los datos correctamente en los grupos a los que les corresponde.

C. GMM

Para este algoritmo se trabajo con 78 dimensiones principales, ya que al trabajar con las 109 mencionadas anteriormente, obtuvimos un overflow error, por lo que todos los experimentos fueron realizados con esa cantidad, esto equivale a más del 90% de varianza de la data. Una vez mencionado esto, se estableció el numero de clusters en 7 (debido a que ya se nos había proporcionado los labels) y el número de iteraciones en 100, para esto se obtuvo la siguiente gráfica del log likelihood.

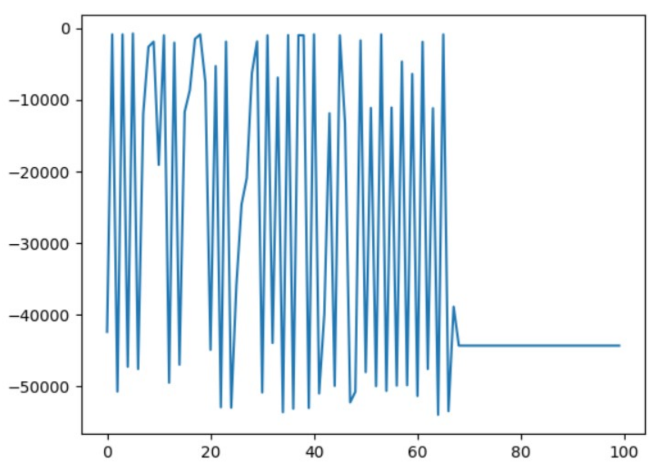

Fig. 5: Log Likelihood vs epocas

Lo que representa un likelihood que no es el deseado ya que converge en un valor bastante bajo concreto, posteriormente se realizo un ***Rand Index*** test, resultando en un valor de 0.1687 lo cual es ínfimo, por lo que se puede concluir que este modelo no es apropiado para la data proporcionada.

Fig. 6: ***Rand index*** para GMM

D. DBSCAN

Para el algoritmo de *DBSCAN* se variaban los parámetros de radio y la cantidad mínima de puntos para generar un *cluster*. Se realizo esto con el objetivo de evaluar el impacto de los dos hiper-parámetros sobre el *dataset* utilizado y evaluar cuál de los dos permitiría generar *clusters* más apropiados.

La primera parte de la experimentación se basa en ingresar diferentes valores del radio y mantener la cantidad mínima de elementos del *cluster* constante en 5 elementos. A continuación, se muestran los *clusters* con los elementos que contienen para las pruebas realizadas:

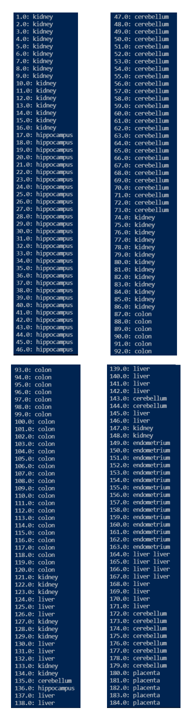

Fig. 7: Cluster con radio 0.5

Fig. 8: Cluster con radio 0.8

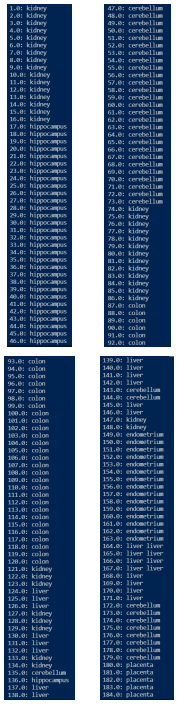

Fig. 9: Cluster con radio 1

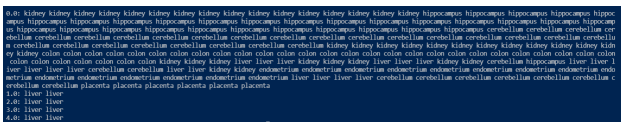

Fig. 10: Cluster con radio 1.000001 parte 3

Estos resultados muestran que para valores mayores a 1, se forman *clusters* menos clusters en los cuáles se agrupan múltiples datos. Esto puede ser una consecuencia de la naturaleza de los datos ya que al ser muy similares el *DBSCAN* es difícil encontrar puntos distantes y que sean distantes para generar apropiadamente los *clusters*. Para el *cluster* con radio mayor 1, se evalúa el *cluster* al verificar el *label* que aparece más y obtener el porcentaje de *labels* que son diferentes. El resultado se muestra a continuación:

Cluster	*Label más común*	Porcentaje de elementos distintos en el *cluster*
0	*Kidney*	78.45%
1	*Liver*	0%
2	*Liver*	0%
3	*Liver*	0%

TABLE I: Tabla de porcentaje de aciertos en el*cluster* de radio mayor a 1

Para el primer *cluster* se encontraban los siguientes elementos y las veces que aparecían:

Elemento	Cantidad de aparciones
kidney	39
hippocampus	31
cerebellum	38
colon	34
liver	18
endometrium	15
placenta	6

La segunda parte del experimento consistió en alterar solamente la cantidad mínima de puntos que debían pertenecer al *cluster*. Se hicieron pruebas para dos valores del radio de búsqueda, el primero fue 1.000001 y el segundo fue 1 para ver si los *clusters* que se generaban se veían modificados de algún modo en ambas medidas.

Los resultados para el radio de búsqueda 1 fueron:

Fig. 11: Cluster con cantidad mínima en 1

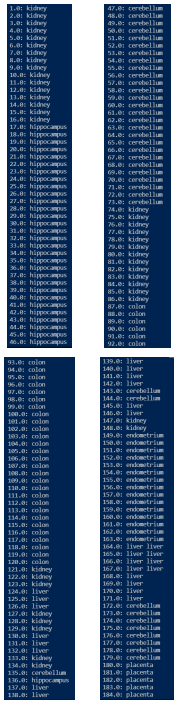

Fig. 12: Cluster con cantidad mínima en 10

Fig. 13: Cluster con cantidad mínima en 50

Los resultados obtenidos al alterar la cantidad mínima de elementos con el radio de búsqueda 1.00001 fueron:

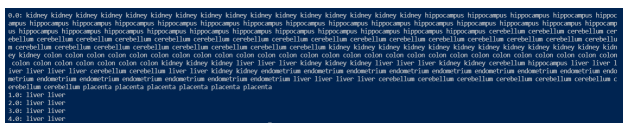

Fig. 14: Cluster con cantidad mínima en 1

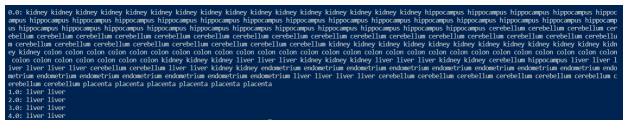

Fig. 15: Cluster con cantidad mínima en 10

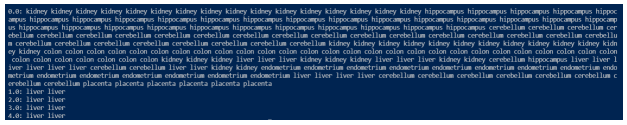

Fig. 16: Cluster con cantidad mínima en 50

Con esto se puede ver que, para este *dataset*, el hiperparámetro de la cantidad mínima de elementos del *cluster* no afecta en gran medida los *clusters* generados por el algoritmo debido a la naturaleza de los elementos del *dataset*. Cabe destacar que para este *dataset*, el radio de búsqueda afecta en gran medida tanto la cantidad de elementos de un *cluster* como la calidad del mismo. Esto se evidencia en los *clusters* que contenían pocos elementos categorizados como ruido y los *clusters* que tenían muy poca precisión al separa los datos. Al usar la métrica de ***Rand index*** para evaluar el desempeño de este modelo, se ve que, cuando se generan *clusters* para cada uno de los valores el comportamiento no nos interesa ya que los *clusters* corresponden en su mayoría a un solo dato. Lo que se debe destacar es que para el radio de búsqueda mayor a 1 entonces podemos mostrar que el valor del ***Rand index*** es 0.010607672034824238; lo que es un valor esperado ya que se han generado imprecisiones al construir los *clusters*.

E. Aglomerative Hierarchical Clustering

Empleamos este algoritmo mediante la librería de sklearn, definiendo los hiperparámetros del número de clusters en 7 y trabajando con 109 dimensiones (95% de varianza), obteniendo un ***Rand index*** de 0.76189.

Fig. 17: ***Rand index*** de AHC

IV. Conclusiones

Una vez realizado las pruebas numéricas podemos concluir que el mejor algoritmo de clustering es K-means, seguido por AGC, DBSCAN para un radio inferior o igual a 1, agrupa la gran mayoría de los puntos en clusters de un elemento por lo que realmente no realiza nada significativo, de la misma manera GMM cae en un underfitting.

Algoritmo	***Rand index***
K-Means	0.770931412135698
GMM	0.1687
DBSCAN	0.010607672034824238
Aglomerative Hierarchical Clustering	0.76189

TABLE II: Tabla de valores de ***Rand index***

REFERENCES

[1] http://www.oranlooney.com/post/ml-from-scratch-part-5-gmm/

[2] Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2021). Mathematics for Machine Learning (English Edition). Cambridge University Press.