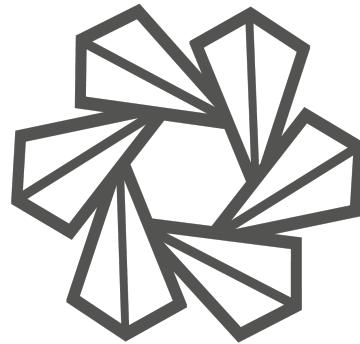
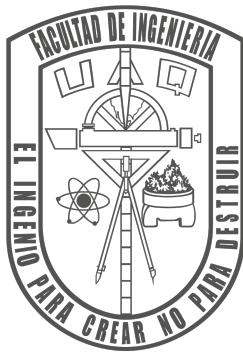
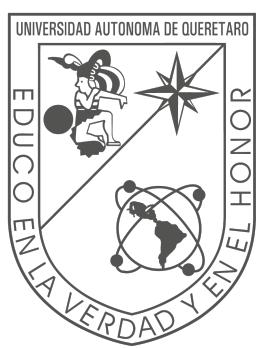


# Universidad Autónoma de Querétaro

Facultad de Ingeniería  
División de Investigación y Posgrado



## Examen 1

K-means y K-means jerárquico para el análisis de datos tabulares en imágenes de radiografías laterales de cráneo

Maestría en Ciencias en Inteligencia Artificial  
Optativa de especialidad IV - Machine Learning

Aldo Cervantes Marquez  
Expediente: 262775  
Profesor: Dr. Marco Antonio Aceves Fernández

Santiago de Querétaro, Querétaro, México  
Semestre 2023-1  
12 de Mayo de 2023

# Índice

<b>1. Objetivo</b>	<b>1</b>
<b>2. Introducción</b>	<b>1</b>
<b>3. Marco Teórico</b>	<b>1</b>
3.1. Métodos de normalización . . . . .	1
3.1.1. Normalización min-max . . . . .	1
3.2. Distancias entre puntos . . . . .	1
3.3. Análisis de Componentes Principales (PCA) . . . . .	2
3.3.1. Eigenvalores y Eigenvectores . . . . .	2
3.4. Agrupamiento K-means . . . . .	4
3.5. Método del codo . . . . .	4
3.6. Agrupamiento jerárquico . . . . .	5
3.6.1. Dendograma . . . . .	7
3.7. Métricas de error . . . . .	7
3.7.1. Coeficiente de Calinski-Harabasz . . . . .	7
3.7.2. Exactitud Entre Modelos . . . . .	8
<b>4. Materiales y Métodos</b>	<b>8</b>
4.1. Materiales . . . . .	8
4.1.1. Base de datos . . . . .	8
4.1.2. Librerías y entorno de desarrollo . . . . .	10
4.2. Metodología . . . . .	10
<b>5. Pseudocódigo</b>	<b>11</b>
<b>6. Resultados</b>	<b>11</b>
6.1. Preprocesamiento y análisis de los datos . . . . .	11
6.2. PCA . . . . .	12
6.3. Método del Codo . . . . .	12
6.4. Dendograma . . . . .	13
6.5. Resultados de los modelos . . . . .	14
6.6. Análisis de agrupamientos . . . . .	15
<b>7. Conclusiones</b>	<b>15</b>
<b>Referencias</b>	<b>16</b>
<b>8. Código Documentado</b>	<b>18</b>

## 1. Objetivo

Esta práctica tiene como objetivo el de aplicar algoritmos de agrupamiento no supervisados, con el fin de observar tendencias o comportamientos que puedan ayudar a mejorar la toma de decisiones, en este caso en datos obtenidos a partir de imágenes. Aplicando la métrica correspondiente para evaluar el modelo.

## 2. Introducción

La práctica consiste en obtener una base de datos tabulares, realizar un análisis de preprocesamiento de datos, identificando valores atípicos, aplicando métodos de imputación, generación de datos sintéticos y normalización. Para posteriormente aplicar el modelo de machine learning de K-means y K-means jerárquico con uso de PCA para analizar los componentes principales y aplicar métricas de evaluación de cada modelo.

## 3. Marco Teórico

Para la realización de la práctica fueron necesarios los siguientes conceptos, los cuales fueron obtenidos principalmente de [1].

### 3.1. Métodos de normalización

Para este tipo de casos, en datos continuos, se utilizó el siguiente método [2, 3].

#### 3.1.1. Normalización min-max

Esta normalización consiste en realizar un escalamiento entre valores definidos  $a$ ,  $b$  y los datos de serán transformadas a ese rango, donde el valor mínimo estará en  $a$  y el máximo en  $b$ . Se calcula mediante la siguiente fórmula:

$$v_{norm} = a + \frac{(v_i - v_{min})(b - a)}{v_{max} - v_{min}} \quad (1)$$

### 3.2. Distancias entre puntos

#### Distancia euclíadiana

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2)$$

#### Distancia Manhattan

$$d(x, y) = \sum_{i=1}^n |x_i - y_i| \quad (3)$$

**Distancia Chebyshev**

$$d(x, y) = \max_{i=1,2,3,\dots,n} |x_i - y_i| \quad (4)$$

**Distancia coseno**

$$d(x, y) = \arccos \left( \frac{x \cdot y}{\|x\| \|y\|} \right) \quad (5)$$

**Distancia euclíadiana normalizada**

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)} \quad (6)$$

Donde  $S^{-1}$  es una matriz con  $\frac{1}{\sigma^2}$  en la diagonal

**Distancia Mahalanobis**

$$d(x, y) = \sqrt{(x - y)^T V^{-1} (x - y)} \quad (7)$$

Donde  $V$  es la matriz de varianzas y covarianzas.

### 3.3. Análisis de Componentes Principales (PCA)

Para realizar el algoritmo de PCA se deben seguir los siguientes pasos:

1. Obtener los datos —en todas sus dimensiones
2. Restar la media. Para que PCA funcione de manera correcta, se requiere restar la media de cada dimensión.
3. Calcular la matriz de covarianza.
4. Calcular los eigenvalores y los eigenvectores de la matriz de covarianza.
5. Escoger los componentes y formar el vector de características. De la columna de los eigenvalores, el elemento con el número mayor es el componente principal del set de datos.
6. En general, una vez que los eigenvectores se encuentran en la matriz de covarianza, se ordenan de mayor a menor dando estos los componentes principales en orden.

#### 3.3.1. Eigenvalores y Eigenvectores

Sea  $V$  un espacio vectorial sobre un campo  $K$  y sea  $T$  una transformación lineal [4, 5]

$$T : V \rightarrow V \quad (8)$$

un vector  $\vec{v}_0$  del espacio vectorial  $V$  se denomina un eigenvector de  $T$  si existe un escalar  $\lambda \in K$  tal que

$$T(\vec{v}_0) = \lambda \vec{v}_0 \quad (9)$$

en este caso  $K$  se denomina el eigenvalor de la transformación  $T$  asociado al eigenvector  $\vec{v}_0$ . De manera semejante podemos definir los eigenvectores y eigenvalores de una matriz  $M$ .

Sea  $M \in K_{n \times n}$  para algún número natural  $n$ , un elemento  $x \in K_n$  se denomina un eigenvector de  $M$  si existe un escalar  $K$  tal que

$$M\vec{x} = \lambda\vec{x} \quad (10)$$

en este caso  $K$  se denomina eigenvalor de la matriz  $M$  asociado al eigenvector  $\vec{x}$ . **Entonces para calcular los eigenvalores y los eigenvectores se debe:**

1. Realizar la operación del determinante de la matriz menos la identidad por lambda:

$$\det(A - \lambda I) \quad (11)$$

donde:

$$\lambda I = \begin{bmatrix} \lambda & 0 & \cdots & 0 \\ 0 & \lambda & \cdots & 0 \\ \vdots & 0 & \lambda & 0 \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}_{n \times n} \quad (12)$$

2. Se hayan las raíces del polinomio formado (resolver para  $\lambda$ )

$$\det(A - \lambda I) = 0 \rightarrow \lambda \quad (13)$$

3. Se calcula el vector propio resolviendo el sistema de ecuaciones para cada eigenvalor

$$(A - \lambda I)v = 0 \quad (14)$$

### Propiedades de los eigenvalores y eigenvectores

1. La traza de la matriz (suma de su diagonal principal) es igual a la suma de todos los valores propios:

$$tr(A) = \sum_{i=1}^n \lambda_i \quad (15)$$

2. El producto de todos los valores propios es igual al determinante de la matriz.

$$\det(A) = \prod_{i=1}^n \lambda_i \quad (16)$$

3. Si existe alguna combinación lineal entre filas o columnas, como mínimo un valor propio de la matriz es igual a 0

### 3.4. Agrupamiento K-means

Este algoritmo permite realizar una clasificación de conjuntos de datos en sus atributos e instancias. Generando grupos con características similares, siendo un algoritmo de aprendizaje no supervisado, en donde no se cuentan con datos etiquetados.

Este algoritmo se basa en la creación de  $n$  centroides que definirán su distancia con los demás datos, generando un grupo. Para realizar el programa se deben seguir los siguientes pasos.

1. Se eligen aleatoriamente los  $k$  objetos que serán los centroides.
2. Se calcula la distancia de cada centroide a cada dato y se reasigna los objetos de acuerdo a la distancia mínima para cada dato.
3. Se recalcula el centroide para que sea el centro geométrico de mi cluster.
4. Repetir etapas 2 y 3 hasta que los centroides no cambien.

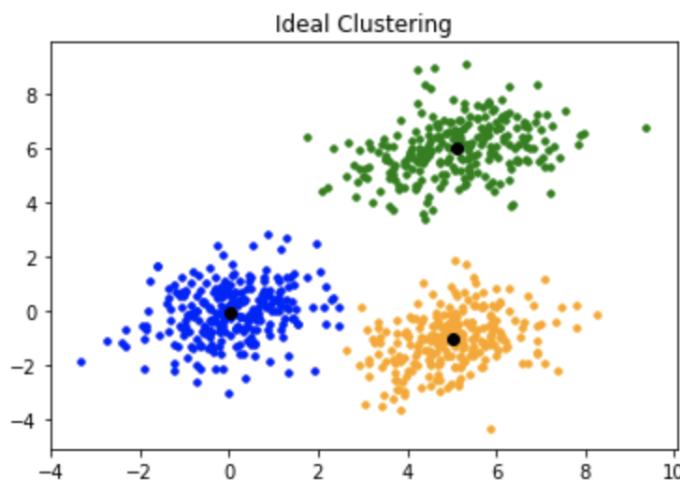


Figura 1: Ejemplo de algoritmo K-means.

### 3.5. Método del codo

Este método permite conocer la cantidad de clusters que serían adecuados para la aplicación, consiste en aplicar la fórmula:

$$I(c_n) = \sum_{i=0}^{n_p} (d(\text{centroid}_{c_n}, p_i)) \approx \sum_{i=0}^{n_p} (p_i - C_n)^2 \quad (17)$$

Tomando en cuenta que: \*  $I(c_n)$  es la inercia del  $n$  cluster \*  $\text{centroid}_{c_n}$  es la coordenada del centroide del  $n$  cluster \*  $p_i$  es un punto que pertenece a la clase  $n$  \*  $n_p$  es el número de puntos que pertenecen a  $n_c$  cluster (es decir  $(p_i \in n_p \in c_n)$ ).

Inercia total:

$$I_t = \frac{\sum_{i_n=0}^n I(i_n)}{n} \quad (18)$$

Generando una gráfica de la siguiente forma:

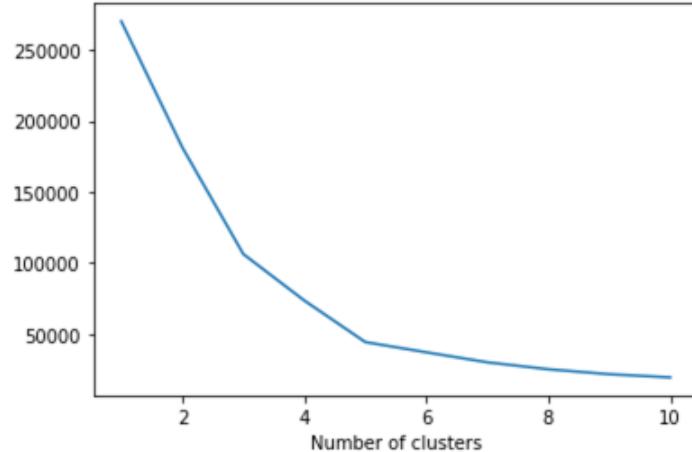


Figura 2: Ejemplo del método del codo.

Observando que el número adecuado de clusters será en donde se vea la mayor inclinación alrededor de su punto, como en ese caso sería entre 3 y 4.

### 3.6. Agrupamiento jerárquico

Este método consiste en modificar el nivel de abstracción de los sub-conjuntos de datos (en matrices de enlace o clusters). Existen básicamente 2 tipos de ordenamiento jerárquico (Figura 3).

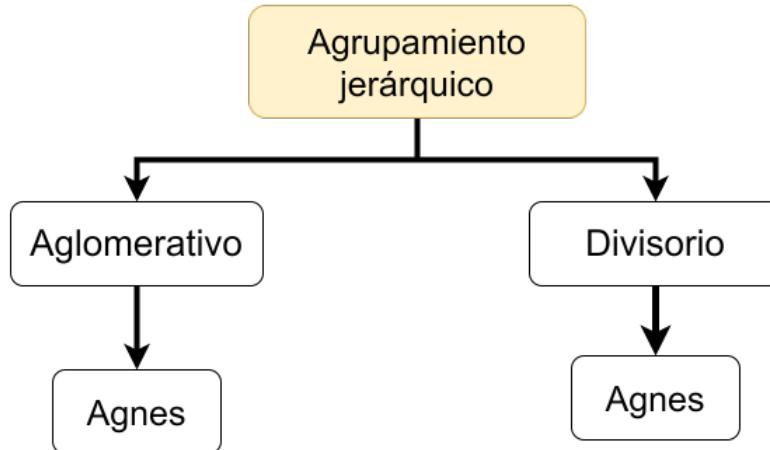


Figura 3: Tipos de agrupamiento jerárquico.

Donde primeramente se encuentra el agrupamiento aglomerativo, el cual consiste en que cada instancia (punto) es un grupo, posteriormente se van realizando mediciones y en donde se encuentre la menor, se junta el grupo y así se siguen uniendo los grupos (Figura 4).

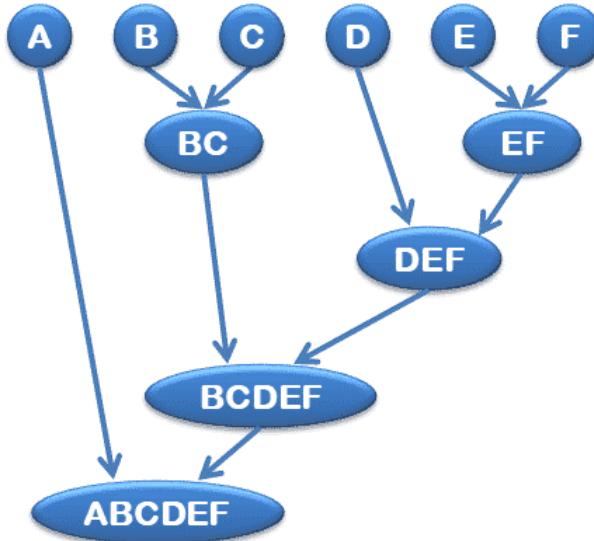


Figura 4: Agrupamiento aglomerativo.

El agrupamiento divisorio es totalmente lo contrario, se parte de un solo cluster y se miden las distancias máximas entre cada uno, anexando un nuevo centroide en la distancia para posteriormente calcular k-means. Dentro de esto, se analizarán 2 métodos que son el *single* y *ward* en AGNES.

El método *single*, para agnes (Agglomerative Nesting) consiste en que el criterio de distancia sea únicamente la menor.

El método *ward*, consiste en la obtención de la varianza mínima entre las mediciones, y de este modo se puede hacer la aglomeración directamente como en *single* u obtener la matriz de enlace que permitirá tener la cantidad de clusters requeridos, mediante un *threshold* que es un umbral de varianza (véase Figura 5) [6].

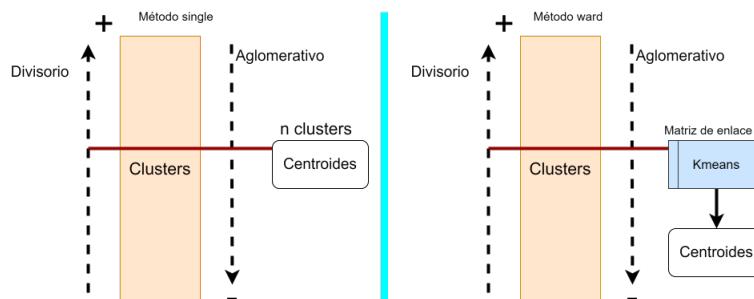


Figura 5: Método single y ward.

### 3.6.1. Dendograma

El dendograma es una representación de como es que las instancias se van agrupando de manera aglomerativa (véase Figura 6), en este caso la cantidad de clusters adecuados serán los que al trazar una linea horizontal tengan la mayor distancia entre la parte más alta y la menor cantidad de cruces con las rayas.

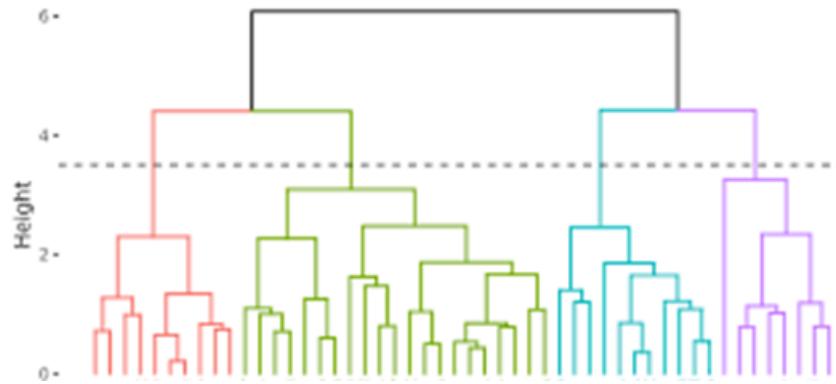


Figura 6: Ejemplo de dendograma.

En este caso se proponen 4 clusters para el conjunto de datos.

## 3.7. Métricas de error

Debido a que no existen datos etiquetados, solo es posible medir la distancia de dispersión y variabilidad entre clusters y la exactitud con respecto a cada uno de los modelos. Por lo que se utilizarán las métricas siguientes [7].

### 3.7.1. Coeficiente de Calinski-Harabatz

Esta métrica consiste en los siguientes aspectos. **Distancia intercluster:**

$$BSSG = \sum_{i=1}^K n_k \times ||C_k - C||^2 \quad (19)$$

donde:

- $n_k$  es el numero de observaciones del cluster  $k$ .
- $C_k$  es el centroide del cluster  $k$ .
- $C$  es el centroide de la base de datos (baricentro).
- $K$  es el número de clusters que hay.

**Distancia intra-cluster**

$$WGSS_k = \sum_{i=1}^{n_k} \|X_{ik} - C_k\|^2 \quad (20)$$

donde:

- $n_k$  es el numero de observaciones del cluster  $k$ .
- $X_{ik}$  es la  $i$  observación del cluster  $k$
- $C_k$  es el centroide del cluster  $k$ .

$$WGSS = \sum_{k=1}^K WGSS_k \quad (21)$$

Entonces el coeficiente de Calinski-Harabasz se obtiene:

$$CH = \frac{BGSS}{WGSS} \times \frac{N - K}{K - 1} \quad (22)$$

donde:

- $N$  es la cantidad total de instancias.
- $K$  es la cantidad total de centroides.

Mientras mayor sea este valor, habrá una mejor distribución de los clusters debida a la varianza entre los mismos.

**3.7.2. Exactitud Entre Modelos**

Se obtiene mediante la analogía de que un modelo es el verdadero y se compara la exactitud con respecto a otro modelo, generando una matriz de exactitud.

$$E_{a,b} = \frac{V_v}{N} \quad (23)$$

donde se suma la cantidad de valores verdaderos (coincidentes) y se va a obtener un valor entre 0 y 1, donde 1 es que coincidió en todas las predicciones. En tanto 0 es lo contrario.

**4. Materiales y Métodos****4.1. Materiales****4.1.1. Base de datos**

La base de datos elegida para esta práctica fue obtenida de [kaggle](#) en donde se realizaron mediciones cefalometricas de una radiografía lateral de cráneo. La base de datos consta de 400 instancias y 38 atributos donde se representan las coordenadas (x,y)de cada elemento importante del cráneo como se muestra a continuación [8]

1. Silla turca
2. Nasion
3. Orbital
4. Porion
5. subespinal
6. Pongonion
7. menton
8. Gnathion
9. Gonion
10. Incisivo inferior
11. Incisivo superior
12. Labio superior
13. Labio inferior
14. Subnasal
15. Tejido suave del menton
16. Espina nasal posterior
17. Espina nasal anterior
18. Articulación

Todo esto se puede observar en la radiografía (véase Figura 7)



Figura 7: Radiografía lateral de cráneo con su respectiva parte.

Se realizó una reducción de atributos, tomando en cuenta distancias más importantes para ortodoncia [9].

Se capturaron las siguientes distancias para la nueva base de datos.

1. Distancia condilo-gonion 19-10
2. Distancia entre dientes superiores e inferiores 11-12
3. Distancia gonion-menton 10-8
4. Distancia nasion-subespinal 2-5
5. Distancia nariz-menton 15-16
6. distancia entre labios 13-14

#### 4.1.2. Librerías y entorno de desarrollo

El análisis de los datos se llevará a cabo en el lenguaje de programación Python dentro del entorno de desarrollo de Jupyter Notebook. Ocupando las librerías `scipy`, `matplotlib`, `seaborn`, `numpy` y `Pandas`.

## 4.2. Metodología

La metodología consiste en la recopilación de las bases de datos, aplicar métodos de preprocesamiento de datos, realizar análisis para conocer la cantidad de clusters que pueden ser requeridos.

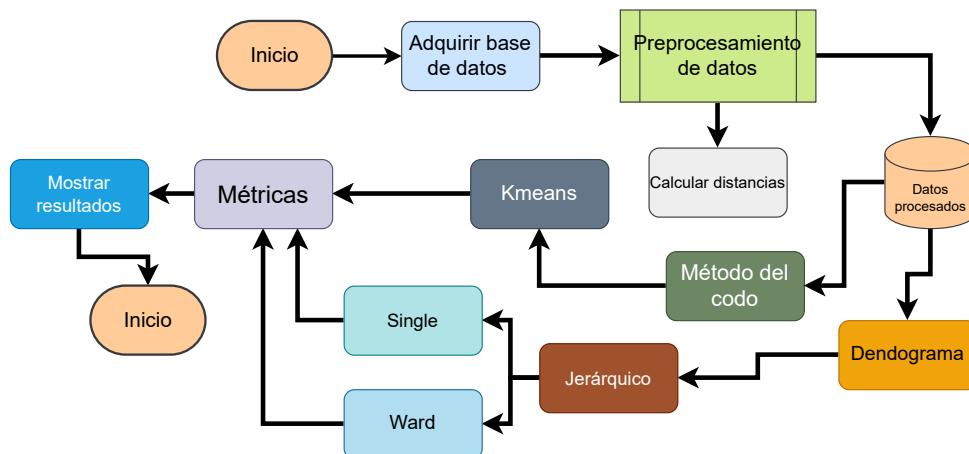


Figura 8: Metodología de la práctica.

## 5. Pseudocódigo

**Algoritmo 1** Pseudocódigo agrupamiento.

```

Inicio
  db
  Class (preprocesamiento):
    Normalización, imputación, generación de datos sintéticos, Normalización
  Class (métricas):
    Calinski, exactitud entre modelos.
  Func (m_codo):
  Func (dendograma)
  Func (k-means):
  Func (jerárquico, tipo):
  Func (PCA):

  db← preprocesamiento(db)
  Para a en num: ## sin pca
    clust=m_codo(db,n)
    k-means(db,clust)
    c_j=dendograma(db)
    jerárquico(db,single,ward)

  db2← PCA(db) ## con PCA
  Regresar ← a con db2
  Mostrar(resultados)

Fin

```

## 6. Resultados

### 6.1. Preprocesamiento y análisis de los datos

Se realizó el cálculo de las distancias mediante el uso de la distancia euclidiana y se observó la distribución de los mismos (véase Figura 9), con el fin de generar nuevos atributos.

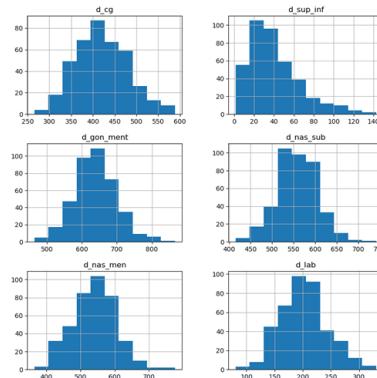


Figura 9: histograma de los atributos.

Donde se observan los datos muy lógicos, pues las distancias suelen tener comportamiento gaussiano y la distancia entre dientes tiene lógica de que sea poco común tener mucho distanciamiento, pues es anatómicamente poco posible.

## 6.2. PCA

Se realizó el análisis de PCA y se obtuvo lo siguiente (véase Tabla 1).

Tabla 1: Resultados de PCA

Distancia condilo-gonion	32.04%
Distancia labial	21.34%
Distancia nariz-mentón	16.03%
Distancia nason-subespinal	12.44%
Distancia gonion-menton	8.89%
Distancia entre dientes	5.22%

Donde se decidió utilizar 4 instancias, pues aportan alrededor del 85 % de los datos.

## 6.3. Método del Codo

Se realizó el método del codo para poder seleccionar el número de clusters necesarios. Se realizó tanto con PCA (véase Figura 11) como sin PCA (véase Figura 10).

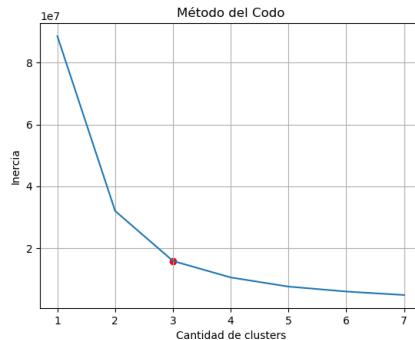


Figura 10: Gráfica del codo sin PCA.

Se observa que entre 2 y 3 puede ser el valor, sin embargo se prefirió tomar 3 clases.

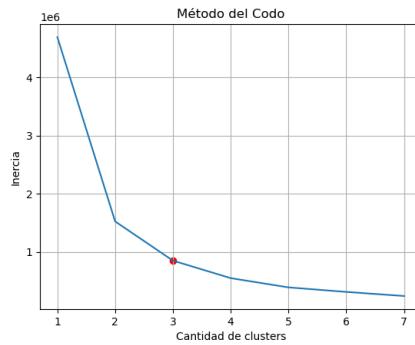


Figura 11: Gráfica del codo con PCA.

Se obtuvieron resultados similares en el comportamiento del codo, por lo que tambien se tomará en cuenta 3 clases.

## 6.4. Dendograma

Se realizó un dendrograma para cada método (single y ward) con y sin PCA, obteniendo los siguientes resultados.

Para el dendrograma con el método single se obtuvo lo siguiente en la Figura 12.

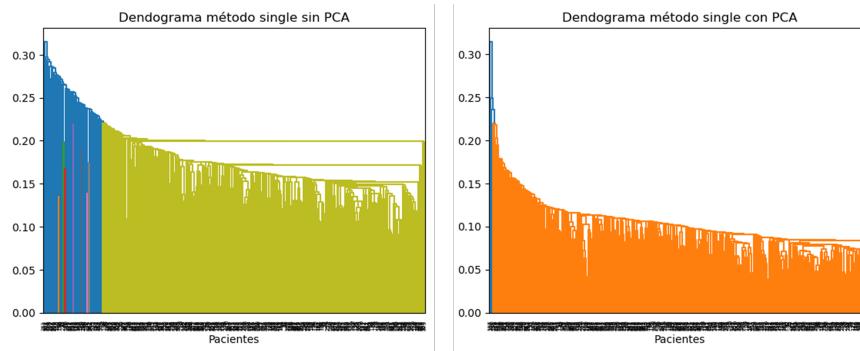


Figura 12: Dendrograma con el método single con y sin PCA.

Se observa que no hay mucha diferencia entre los métodos, por lo que es complicado decidir un valor exacto de clusters, por lo que se tuvo que experimentar y con 3 clusters se tenia una división algo densa entre los datos, esto significa que casi todo un cluster tiene casi todos los datos.

Para el dendrograma con el método ward se obtuvo lo siguiente en la Figura 13.

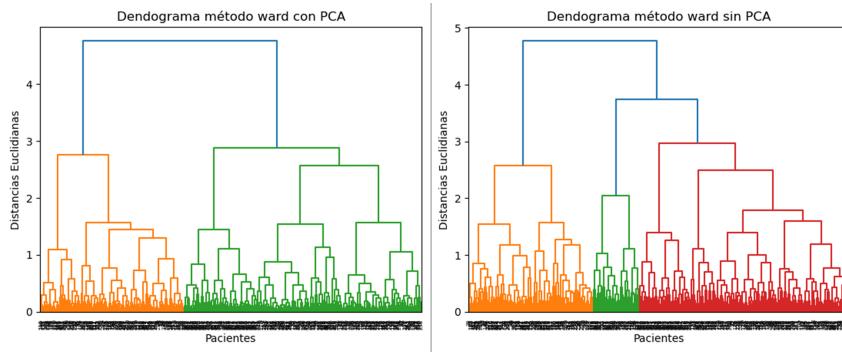


Figura 13: Dendograma con el método ward con y sin PCA.

Se observa que esta mejor organizado el dendograma y se puede ver que entre 3 y 4 grupos es la mejor opción para el modelo.

## 6.5. Resultados de los modelos

Para poder evaluar los modelos, se aplicó el coeficiente de Calinski obteniendo los siguientes resultados en la Tabla 2

Tabla 2: Resultados de métricas del coeficiente de Calinski

K-means	2.72E+11
K-means PCA	8.82E+10
K-means jerárquico single	2.84E+08
K-means jerárquico single PCA	2.30E+10
K-means jerárquico ward	1.98E+11
K-means jerárquico ward PCA	1.12E+11

Como se observa, son cantidades muy grandes, lo que en principio es bueno, pues indica una mayor dispersion entre clusters, sin embargo, hay que tomar en cuenta que no se encuentran normalizadas y es difícil decir con exactitud cual es el mejor modelo, para eso se tiene la métrica de la exactitud entre los modelos, obteniendo la Figura 15.

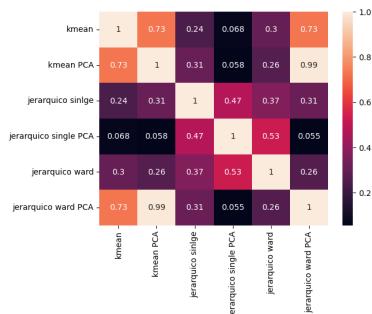


Figura 14: Exactitud entre modelos.

Se logra apreciar que hubo modelos que no pudieron funcionar del todo bien pero hubo algunas que si predijeron lo que había predicho otro modelo. Obteniendo en porcentajes lo siguiente (Tabla 3):

Tabla 3: Medias de desempeño de exactitud en cada modelo

K-means	0.414
K-means PCA	0.4695
K-means jerárquico single	0.3405
K-means jerárquico single PCA	0.237
K-means jerárquico ward	0.3445
K-means jerárquico ward PCA	0.4695

## 6.6. Análisis de agrupamientos

Se separó una muestra de las 3 diferentes clases obtenidas. Por lo que podemos observar lo siguiente:



Figura 15: Radiografía lateral de cráneo en varias clases.

La clase 0 se relaciona con mandíbulas pequeñas y facciones suaves, además de tener una distancia media entre labios y la mordida abierta (los dientes están muy angulados hacia la boca).

La clase 1 se observa que tiene una mandíbula estándar y facciones medias, se podría decir que tiene facciones algo alargadas pero buenas para un tratamiento de ortodoncia.

La clase 2 se observa que tiene un prognatismo muy remarcado y sus distancias a pesar de ser cercanas a lo normal, se observa que tiene una clase canina 3.

## 7. Conclusiones

La presente práctica mostró la aplicación de los modelos de clasificación con K-means y un extra que fue la clasificación jerárquica. Etiquetando datos no etiquetados, aplicando PCA para observar

los componentes principales de la base de datos, reducir la dimensionalidad y evaluar las métricas de desempeño.

Se observó que fue posible clasificar en 3 grupos los datos de las radiografías, obteniendo que hay personas mas susceptibles a ciertos tratamientos ortodonticos que otras, lo que conlleva caminos diferentes para el especialista llevar el tratamiento.

Obteniendo las siguientes conclusiones sobre el trabajo desarrollado

- Los algoritmos de k-means son bastante rápidos en comparación a los jerarquicos, puesto que se tomaron aproximadamente 2 minutos en correr y aparte k-means para generar centroides en la matriz.
- Los algoritmos jerarquicos requieren de métodos más robustos de procesamiento de datos y de solución como lo fue el método ward.
- El uso de estos algoritmos es de bastante utilidad para su uso en problemas con datos no etiquetados.
- Las métricas utilizadas dan un panorama de como es que el modelo se desempeña, además de poder observar su desempeño ante otros modelos.
- El método jerarquico single necesita de menor cantidad de datos y atributos para poder funcionar de una manera más correcta, pues es el algoritmo más sencillo de jerarquía aglomerativa.
- Este modelo puede ser un precursor para el análisis clínico en tratamientos de ortodoncia, obteniendo resultados tangibles que pueden obtener características interesantes en las medidas de una radiografía.

## Referencias

- [1] M. Fernández, *Inteligencia Artificial para Programadores con Prisa*. Amazon Digital Services LLC - KDP Print US, 2021.
- [2] S. Lasse, “Data augmentation for tabular data.” <https://medium.com/analytics-vidhya/data-augmentation-for-tabular-data-f75c94398c3e>, November 2021. (Accessed on 03/16/2023).
- [3] L. Al Shalabi and Z. Shaaban, “Normalization as a preprocessing engine for data mining and the approach of preference matrix,” in *2006 International Conference on Dependability of Computer Systems*, pp. 207–214, 2006.
- [4] J. Ortega, “Eigenvalores y eigenvectores.” <http://gmc.geofisica.unam.mx/papime2020/index.php/articulos/38-eigenvalores-y-eigenvectores#:~:text=De%20manera%20semejante%20podemos%20definir,eigenvalores%20de%20una%20matriz%20M.&text=M%E2%86%92x%3D%CE%BB%E2%86%92,asociado%20al%20eigenvector%20%E2%86%92x%20.> (Accessed on 05/11/2023).

- [5] “Calcular los valores propios y vectores propios de una matriz.” <https://www.matricesydeteminantes.com/matrices/calcular-valores-propios-autovalores-y-vectores-propios-autovectores-de-una-matriz/>. (Accessed on 05/11/2023).
- [6] “14.7 - ward’s method — stat 505.” <https://online.stat.psu.edu/stat505/lesson/14/14.7>. (Accessed on 05/11/2023).
- [7] “Calinski-harabasz index for k-means clustering evaluation using python.” <https://pyshark.com/calinski-harabasz-index-for-k-means-clustering-evaluation-using-python/>. (Accessed on 05/11/2023).
- [8] L. Claudia and C.-W. W. ant et.al, “Fully automatic system for accurate localisation and analysis of cephalometric landmarks in lateral cephalograms — scientific reports.” <https://www.nature.com/articles/srep33581>. (Accessed on 05/11/2023).
- [9] K. Toshio, “Cefalometrix.” <https://cefalometrix.blogia.com/>, 2005. (Accessed on 05/11/2023).

# Examen 1 clasificación y PCA

May 12, 2023

## 0.1 Clasificación y PCA

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

[2]: d_ceph=pd.read_csv('train_senior.csv')
d_ceph2=pd.read_csv('test1_senior.csv')
d_ceph3=pd.read_csv('test2_senior.csv')

[3]: df=pd.concat([d_ceph,d_ceph2])
df=pd.concat([df,d_ceph3])
df=df.drop(['image_path'],axis=1)
datab=df.copy()
datab=datab.to_numpy().astype(float)
datab.shape

[3]: (400, 38)
```

## 1 Preprocesamiento de los datos.

```
[4]: def d_p(a,b):
    x1=a.iloc[:,0]
    y1=a.iloc[:,1]
    x2=b.iloc[:,0]
    y2=b.iloc[:,1]
    return np.sqrt((x1-x2)**2+(y1-y2)**2)
#return np.linalg.norm(a-b)

[5]: #distancia condilion-golion 19-10
d_cg=d_p(df[['19_x','19_y']].astype(float),df[['10_x','10_y']].astype(float))
# Distancia entre dientes superiores e inferiores 11-12
d_sup_inf=d_p(df[['11_x','11_y']].astype(float),df[['12_x','12_y']].astype(float))
# Distancia de gonion a menton 10-8
d_gon_menton=d_p(df[['10_x','10_y']].astype(float),df[['8_x','8_y']].astype(float))
```

```
# distancia de nasion a sub espinal 2-5
d_nas_sub=d_p(df[['2_x','2_y']].astype(float),df[['5_x','5_y']].astype(float))
# Distancia nariz-menton 15-16
d_nas_men=d_p(df[['15_x','15_y']].astype(float),df[['16_x','16_y']].
    .astype(float))
# Distancia labios 13-14
d_lab=d_p(df[['13_x','13_y']].astype(float),df[['14_x','14_y']].astype(float))
```

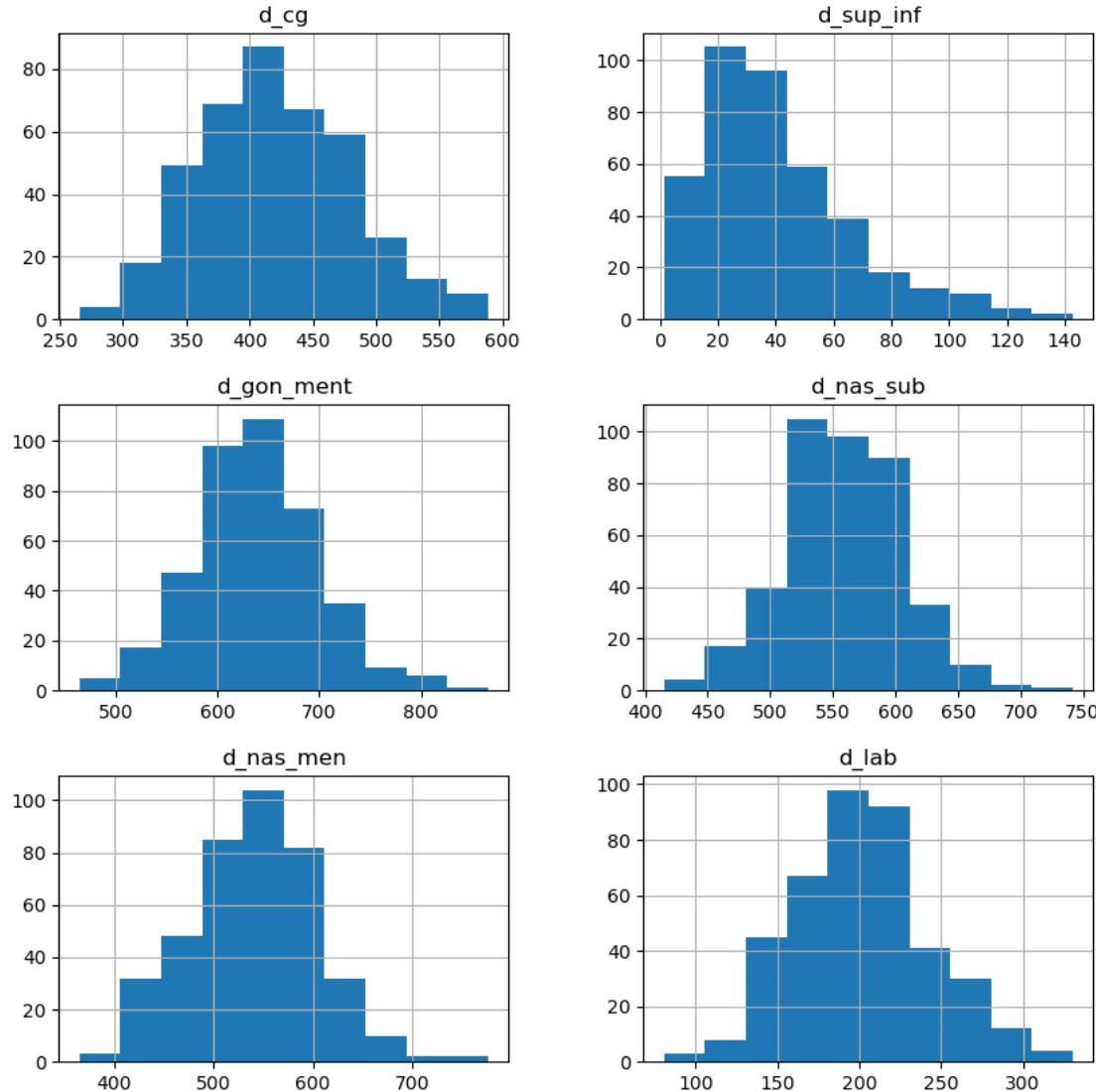
```
[6]: data_b=pd.concat([d_cg,d_sup_inf,d_gon_ment,d_nas_sub,d_nas_men,d_lab],axis=1)
data_b.columns=['d_cg','d_sup_inf','d_gon_ment','d_nas_sub','d_nas_men','d_lab']

fig = plt.figure(figsize = (10,10))
ax = fig.gca()
data_b.hist(ax = ax)
#data_b.hist()
```

C:\Users\aldoa\AppData\Local\Temp\ipykernel\_8852\2721853312.py:6: UserWarning:  
To output multiple subplots, the figure containing the passed axes is being  
cleared.

```
    data_b.hist(ax = ax)
```

```
[6]: array([[<AxesSubplot:title={'center':'d_cg'}>,
    <AxesSubplot:title={'center':'d_sup_inf'}>],
    [<AxesSubplot:title={'center':'d_gon_ment'}>,
    <AxesSubplot:title={'center':'d_nas_sub'}>],
    [<AxesSubplot:title={'center':'d_nas_men'}>,
    <AxesSubplot:title={'center':'d_lab'}>]], dtype=object)
```



## 2 PCA

Para realizar el algoritmo de PCA se deben seguir los siguientes pasos:

1. Obtener los datos —en todas sus dimensiones—
2. Restar la media. Para que PCA funcione de manera correcta, se requiere restar la media de cada dimensión.
3. Calcular la matriz de covarianza.
4. Calcular los eigenvalores y los eigenvectores de la matriz de covarianza.
5. Escoger los componentes y formar el vector de características. De la columna de los eigenvalores, el elemento con el número mayor es el componente principal del set de datos.
6. En general, una vez que los eigenvectores se encuentran en la matriz de covarianza, se ordenan de mayor a menor dando estos los componentes principales en orden.

- Calcular eigenvalores y eigenvectores [fuente eigenvalores](#) además de [eigenvalores y eigenvectores 2](#).

Sea  $V$  un espacio vectorial sobre un campo  $K$  y sea  $T$  una transformación lineal

$$T : V \rightarrow V$$

un vector  $\vec{v}_0$  del espacio vectorial  $V$  se denomina un eigenvector de  $T$  si existe un escalar  $\lambda K$  tal que

$$T(\vec{v}_0) = \lambda \vec{v}_0$$

en este caso  $\lambda K$  se denomina el eigenvalor de la transformación  $T$  asociado al eigenvector  $\vec{v}_0$

De manera semejante podemos definir los eigenvectores y eigenvalores de una matriz  $M$ .

Sea  $MK_{n \times n}$  para algún número natural  $n$ , un elemento  $x \in K_n$  se denomina un eigenvector de  $M$  si existe un escalar  $\lambda K$  tal que

$$M\vec{x} = \lambda \vec{x}$$

en este caso  $\lambda K$  se denomina eigenvalor de la matriz  $M$  asociado al eigenvector  $\vec{x}$ .

**Entonces para calcular los eigenvalores y los eigenvectores se debe:** 1. Realizar la operación del determinante de la matriz menos la identidad por lambda:

$$\det(A - \lambda I)$$

donde:

$$\lambda I = \begin{bmatrix} \lambda & 0 & \cdots & 0 \\ 0 & \lambda & \cdots & 0 \\ \vdots & 0 & \lambda & 0 \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}_{n \times n}$$

2. Se hayan las raíces del polinomio formado (resolver para  $\lambda$ )

$$\det(A - \lambda I) = 0 \rightarrow \lambda$$

3. Se calcula el vector propio resolviendo el sistema de ecuaciones para cada eigenvalor

$$(A - \lambda I)v = 0$$

**Propiedades de los eigenvalores y eigenvectores** 1. La traza de la matriz (suma de su diagonal principal) es igual a la suma de todos los valores propios:

$$tr(A) = \sum_{i=1}^n \lambda_i$$

2. El producto de todos los valores propios es igual al determinante de la matriz.

$$\det(A) = \prod_{i=1}^n \lambda_i$$

3. Si existe alguna combinación lineal entre filas o columnas, como mínimo un valor propio de la matriz es igual a 0

```
[7]: def covarianza(X,Y):
    x_mean = X.mean()
    y_mean = Y.mean()
    n = len(X)
    cov = (((X-x_mean)*(Y-y_mean)).sum())/(n-1)
    return cov

def matriz_cov(data):
    atributos = data.columns
    n = len(atributos)
    m = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            X = data[atributos[i]]
            Y = data[atributos[j]]
            m[i][j] = covarianza(X,Y)
    return m

def pca1(data): # Ya tiene que estar normalizado
    atributos = list(data.columns)
    n = len(atributos)
    for a in atributos:
        data[a]=data[a]-data[a].mean()
    #m_cov=matriz_cov(data)
    m_cov=np.cov(np.transpose(data))
    sns.heatmap(m_cov,annot=True)
    eival,eivec=np.linalg.eig(m_cov)
    p=(eival/eival.sum())*100
    return p,eivec

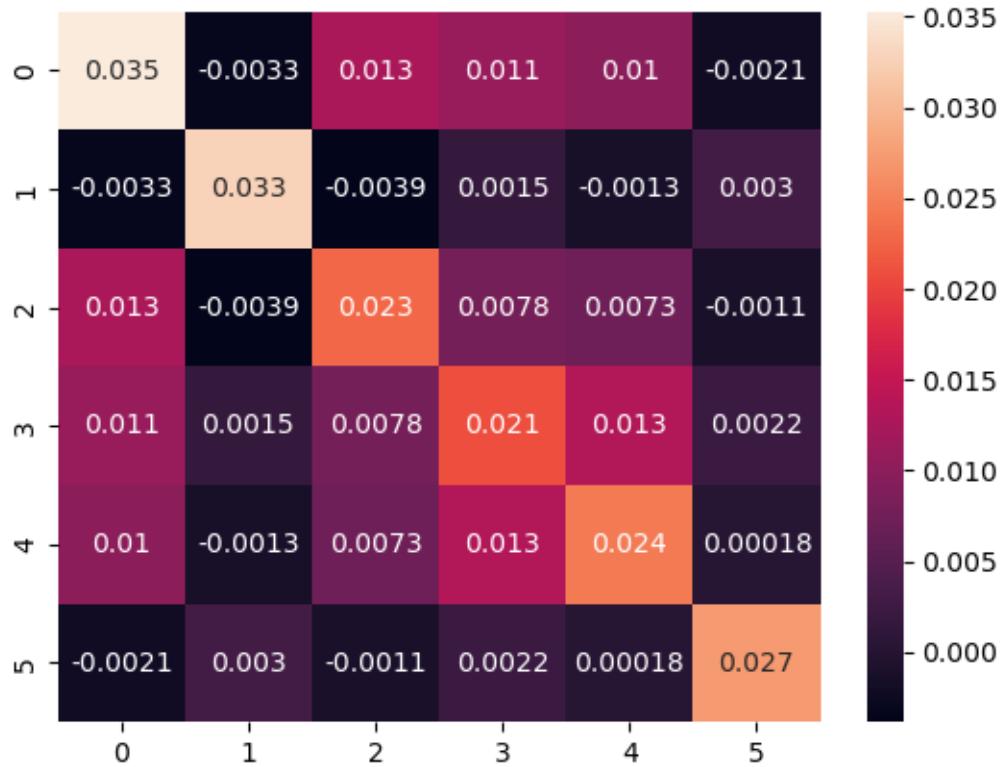
def norm_min_max(x,a,b): # Para todo el DataFrame
    l=list(x.columns)
    v_max=0
    v_min=0
    res=pd.DataFrame()
    for val in l:
        v_max=x[val].max()
        v_min=x[val].min()
        r_dt=v_max-v_min
        r_norm=b-a
        res[val]=(x[val]-v_min)/r_dt+a
    return res
```

```
d=x[val]-v_min
dpct=d/r_dt
dnorm=r_norm*dpct
data=a+dnorm
aa=pd.DataFrame(data,columns=[val])
res[val]=data
return res

def ordenar(porcentajes,columnas):
    n=len(columnas)
    p2=porcentajes.copy()
    porcentaje=np.sort(porcentajes)[::-1]
    dicti={}
    for a in range(n):
        ub=int(np.where(porcentaje[a]==p2)[0])
        dicti[columnas[ub]]=porcentaje[a]
    return dicti
```

```
[8]: porcentaje,eivec=pca1(norm_min_max(data_b,0,1))
#np.cov(np.transpose(data_b)).shape
por=ordenar(porcentaje,data_b.columns)
print(por)
```

```
{'d_cg': 36.04072130342554, 'd_lab': 21.34712266379575, 'd_nas_men': 16.032413373957308, 'd_nas_sub': 12.442237832788472, 'd_gon_ment': 8.892535879862113, 'd_sup_inf': 5.244968946170824}
```



[ ]:

### 3 K-Means

1. Se eligen aleatoriamente los k objetos que serán los centroides.
2. se calcula la distancia de cada centroide a cada dato y se reasigna los objetos de acuerdo a la distancia mínima para cada dato.
3. Se recalcula el centroide para que sea el centro geométrico de mi cluster
4. repetir etapas 2 y 3 hasta que los centroides no cambien.

#### 3.1 Distancias

##### Euclidiana

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

##### Manhattan

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

**Chebyshev**

$$d(x, y) = \max_{i=1,2,3,\dots,n} |x_i - y_i|$$

**Coseno**

$$d(x, y) = \arccos \left( \frac{x \cdot y}{\|x\| \|y\|} \right)$$

**Distancia euclíadiana normalizada**

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

Donde  $S^{-1}$  es una matriz con  $\frac{1}{\sigma}$  en la diagonal

**D. Mahalanobis**

$$d(x, y) = \sqrt{(x - y)^T V^{-1} (x - y)}$$

Donde  $V$  es la matriz de varianzas y covarianzas

```
[9]: def d_euc(a,b):
    return np.linalg.norm(a-b)

def d_manhattan(a,b):
    return np.sum(np.abs(a-b))

def k_means(data,n_c,distancia):

    cls=np.random.choice(data.shape[0],n_c,replace=False)
    clusters=data[cls,:]
    #print('clusters iniciales',clusters)
    ren,col=data.shape
    distancias=np.zeros((n_c,ren,1))
    d_ant=np.ones((n_c,ren,1))
    pert=np.zeros((ren,1))
    n_centroide=np.zeros((n_c,col))
    #print(n_centroide[0,:])
    cent=n_centroide+1
    iteracion=0
    #print('-----')
    #while not np.array_equal(cent,n_centroide) or iteracion<10: ##
    while iteracion<20 and not np.array_equal(cent,clusters):
        cent=np.copy(clusters)

        n_centroide=np.zeros((n_c,col))
        for ib in range(n_c):
            for ia in range(ren): # se calculan las distancias
                distancias[ib,ia,0]=distancia(clusters[ib,:],data[ia,:])
            for ic in range(ren): # Observar el valor más cercano
                if distancias[ib,ia,0]>distancias[ib,ic,0]:
                    cent[ib,:]=clusters[ic,:]
                    distancias[ib,ia,0]=distancias[ib,ic,0]
        iteracion+=1
    return cent
```

```

pert[ic,0]=np.where(distancias[:,ic,:]==min(distancias[:,ic,::
→]))[0][0]
pert=pert.astype(int)
#print(pert)
for idd in range(ren):
    #print(pert[idd,0])
    n_centroide[pert[idd,0],:]+=data[idd,:]
for abb in range(n_c):
    clusters[abb,:]=n_centroide[abb,:]/(np.count_nonzero(pert==abb))

iteracion+=1

#print(iteracion)
return clusters,pert

```

## 4 Método del codo (elbow method)

Formula para la inercia en cada cluster:

$$I(c_n) = \sum_{i=0}^{n_p} (d(\text{centroid}_{c_n}, p_i))^2 \approx \sum_{i=0}^{n_p} (p_i - C_n)^2$$

Tomando en cuenta que:  
\*  $I(c_n)$  es la inercia del  $n$  cluster  
\*  $\text{centroid}_{c_n}$  es la coordenada del centroide del  $n$  cluster

\*  $p_i$  es un punto que pertenece a la clase  $n$   
\*  $n_p$  es el número de puntos que pertenecen a  $n_c$  cluster (es decir  $(p_i \in n_p \in c_n)$ ) Inercia total:

$$I_t = \frac{\sum_{i_n=0}^n I(i_n)}{n}$$

Donde:

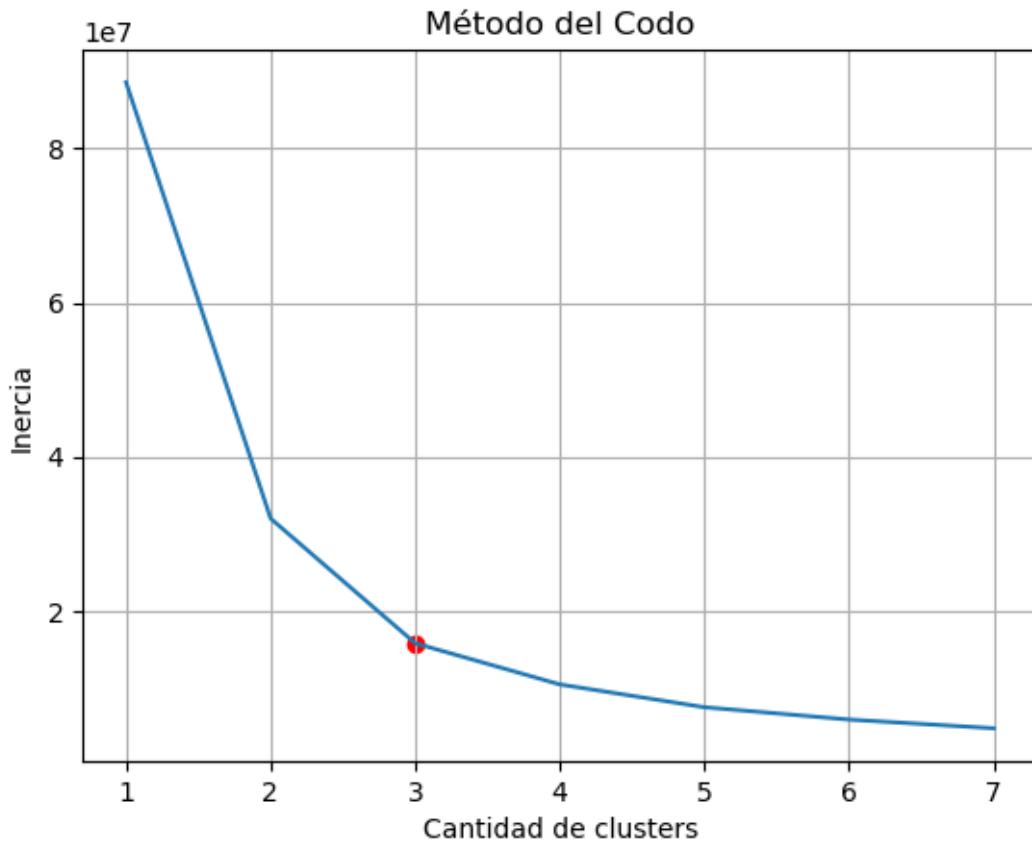
- $I(i_n)$  es la inercia de cada cluster.
- $n$  es la cantidad de clusters que hay

```
[10]: def e_met(data,c_max,distancia):
    res=[]
    n_clus=range(1,c_max+1)
    temp=0
    ren,col=data.shape
    for a in range(c_max):
        r_sum=np.zeros((a+1,1))
        cluster,pert=k_means(data,a+1,distancia)
        for b in range(a+1):
            dt=np.where(pert==b)[0] #Grupo de cada dato
            for c in range(len(dt)):
                r_sum[b]+=distancia(cluster[b,:],data[dt[c],:])**2
```

```
#r_sum[b]/=len(dt)
temp=float(sum(r_sum[:]))/(a+1)
res.append(temp)
temp=0
plt.plot(n_clus,res)
plt.grid()
plt.xticks(n_clus)
plt.xlabel('Cantidad de clusters')
plt.ylabel('Inercia')
plt.title('Método del Codo')
return res,n_clus
```

```
[12]: valor=3
inercia,n_cl=e_met(datab,7,d_euc)
plt.scatter(valor,inercia[valor-1],color='red')
```

```
[12]: <matplotlib.collections.PathCollection at 0x1f7f2ddb7f0>
```



```
[13]: res,lab=k_means(data_b.to_numpy(),3,d_euc)
res
```

```
[13]: array([[369.03546555, 47.20840588, 588.51900474, 516.61137185,
   471.3387733 , 212.8237487 ],
 [410.62633622, 37.82717092, 634.64592727, 555.77892748,
 542.90363175, 198.28446429],
 [486.98550288, 38.5829431 , 691.78747275, 601.08058458,
 599.95532171, 199.86543944]])
```

[14]: lab















```
[0],  
[0],  
[0],  
[1],  
[1],  
[1],  
[0],  
[1],  
[0],  
[2],  
[1],  
[0],  
[0],  
[2],  
[1],  
[1],  
[0],  
[1],  
[1],  
[0],  
[1],  
[1],  
[2],  
[0],  
[1],  
[2],  
[0],  
[1],  
[0],  
[0],  
[0],  
[0],  
[0],  
[2]))
```

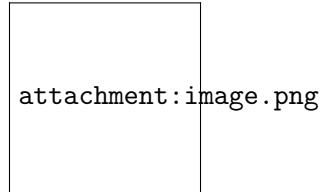
## 5 Clustering jerárquico

El algoritmo de clúster jerárquico agrupa los datos basándose en la distancia entre cada uno y buscando que los datos que están dentro de un clúster sean los más similares entre sí.

Existen dos tipos de clustering, el basado en divisiones y el aglomerativo. Como se puede observar en el dendograma.



El basado en divisiones consiste en



```
[15]: def agnes(data,distancia): #aglomerativo distancia mínima con k-means
    # partimos de que todos los puntos son un cluster diferente.
    d2=np.copy(data)
    d3=np.copy(data)
    ren,col=d2.shape
    n_cl=ren
    centroides=[]
    centroides.append(d2)
    distancias=np.zeros((n_cl,ren,1))
    print('inicio')
    while n_cl >1:
        distancias=np.zeros((n_cl,ren))
        for a in range(n_cl):
            for b in range(ren):
                if a==b:
                    distancias[a,b]=10000
                else:
                    distancias[a,b]=distancia(centroides[-1][a,:],d3[b,:])
    #mascara=np.eye(distancias.shape[0],dtype=bool)
    #dist_n=np.ma.array(distancias,mask=mascara)

    p_min=np.unravel_index(np.argmin(distancias),distancias.shape)
    print('coord min',p_min) # checar valor
    #n_cl-=1
    # obtener promedio
    n_cluster=(d3[a,:]+d3[b,:])/2
    #print(np.transpose(n_cluster).shape)

    # eliminar clusters
    #print(distancias.shape)
    d3=np.delete(d3,(p_min[0],p_min[1]),axis=0)
    print(d3.shape)
```

```

d3=np.vstack((d3,n_cluster))
centroides.append(d3)
ren2,col2=centroides[-1].shape
n_cl=ren2
distancias=np.zeros((n_cl,ren))
ren,col=d3.shape
return centroides

```

[ ]:

```

[16]: ## Se tienen que ir haciendo las categorias primero y despues se van haciendo
       ↵ las mediciones
def agnesb(data,distancia): #aglomerativo distancia minima con k-means
    # partimos de que todos los puntos son un cluster diferente.
    d2=np.copy(data)
    d3=np.copy(data)
    ren,col=d2.shape
    n_cl=ren
    centroides=[]
    #centroides.append(label)
    distancias=np.zeros((n_cl,ren,1))
    label=np.arange(ren).reshape(-1,1)
    #print(label)
    centroides.append(label)
    print('inicio')
    distancias=np.zeros((n_cl,ren))
    for a in range(n_cl):
        for b in range(ren):
            if a==b:
                distancias[a,b]=10000
            else:
                distancias[a,b]=distancia(d3[a,:],d3[b,:])
    #mascara=np.eye(distancias.shape[0],dtype=bool)
    #dist_n=np.ma.array(distancias,mask=mascara)
    d_ord=np.sort(distancias,axis=None)
    #print(d_ord.shape)
    r=d_ord.shape
    for xx in range(r[0]):
        if d_ord[xx]==10000:
            pass
        else:
            p_min=np.where(d_ord[xx]==distancias)
            l2=len(np.unique(label))
            if p_min[0][0]==p_min[1][0]:
                pass
            #print(p_min)
            else:

```

```

        label=np.where(label==p_min[0][0],p_min[1][0],label)
        #print(len(np.unique(label)))
        if len(np.unique(label))!=12:
            centroides.append(label)
    #label=np.where(label==label[0],label[0],label)
    return centroides

```

```
[17]: import time
t1=time.time()
cent=agnesb(data_b,d_euc)
t2=time.time()
print('tiempo: ',t2-t1)
```

inicio  
tiempo: 70.71974444389343

```
[18]: print(len(cent))
val=cent[-3]
valores_unicos, frecuencias = np.unique(val, return_counts=True)

# Imprimir los valores y sus frecuencias
for valor, frecuencia in zip(valores_unicos, frecuencias):
    print("Valor:", valor, "- Frecuencia:", frecuencia)
```

400  
Valor: 386 - Frecuencia: 2  
Valor: 392 - Frecuencia: 2  
Valor: 399 - Frecuencia: 396

```
[19]: def ajustar(clusters,data,distancia):
    n_clus,atrib=clusters.shape
    ren,col=data.shape
    pert=np.zeros((ren,1))
    d_clus=np.zeros((1,n_clus))
    for a in range(ren):
        for b in range(n_clus):
            d_clus[0,b]=distancia(clusters[b,:],data[a,:])
            #print(d_clus[0,b])
        #print(d_clus)
        pert[a,0]=np.where(d_clus==d_clus.min())[1][0]
        d_clus=np.zeros((1,n_clus))
        #print('fin')
        pert=pert.astype(int)
    return pert
```

```
[20]: def calc_centroide(labels,data,distancia):
    v=len(np.unique(labels))
    ren,col=data.shape
```

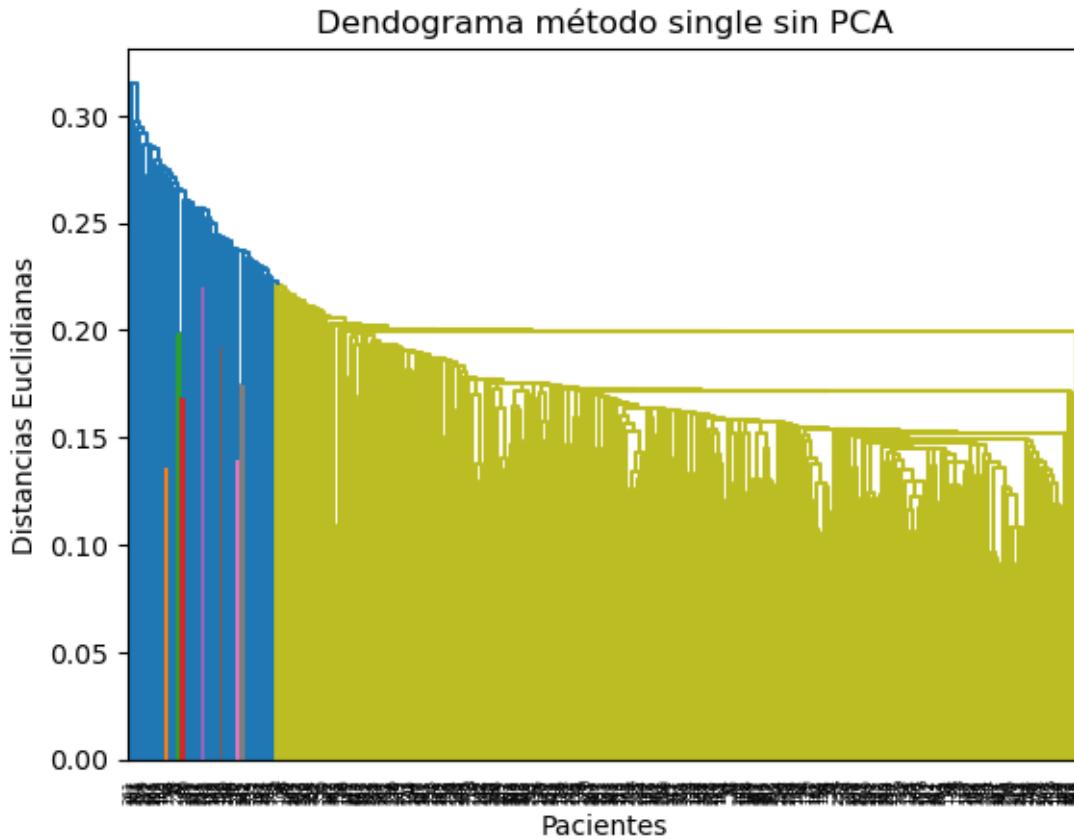
```
centroide=np.zeros((v,col))
valores_unicos, frecuencias = np.unique(labels, return_counts=True)
temp=0
print(valores_unicos)
print(frecuencias)
# Imprimir los valores y sus frecuencias
for a in range(len(valores_unicos)):
    for b in range(ren):
        if int(labels[b])==valores_unicos[a]:
            #centroide[a,:]+=data[b,:]
            centroide[a,:]=centroide[a,:]+data[b,:]
    centroide[a,:]/=frecuencias[a]
return centroide
```

```
[21]: cent_jerarquico=calc_centroide(cent[-3],data_b.to_numpy(),d_euc)
print(cent_jerarquico)
```

```
[386 392 399]
[ 2  2 396]
[[435.26355571 35.97343131 682.10702496 724.39561484 762.82250939
 242.27138334]
 [354.88562042 22.32740514 478.75114501 439.2303629 430.70200065
 97.84635199]
 [420.93030928 40.4387363 639.07378783 557.94917736 539.92052193
 202.59086738]]
```

```
[68]: import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(norm_min_max(data_b,0,1), method = 'single'))

plt.title('Dendograma método single sin PCA')
plt.xlabel('Pacientes')
plt.ylabel('Distancias Euclidianas')
plt.show()
```



## 6 Métricas de evaluación.

calinski coefficient 1. intercluster dispersión

$$BSGG = \sum_{i=1}^K n_k \times \|C_k - C\|^2$$

donde:

```
[23]: def bgss(data,centroids,frec):
    nc,col2=centroids.shape
    ren,col=data.shape
    temp=0
    baricentro=np.sum(data)/ren
    for a in range(nc):
        temp+=d_euc(centroids[a,:],baricentro)**2
        temp*=frec[a]
    return temp

def wgss(data, labels, centroid):
```

```

num_clusters = len(np.unique(labels))
total_dispersion = 0.0

for cluster in range(num_clusters):
    cluster_data = data[labels == cluster]
    cluster_centroid = centroid[cluster]

    # Calcular la distancia Euclíadiana entre cada punto del cluster y el centroide
    distances = np.linalg.norm(cluster_data - cluster_centroid, axis=1)

    # Sumar las distancias al total de dispersión intra-cluster
    total_dispersion += np.sum(distances)

return total_dispersion

def calinski(v1,v2,data,clusters):
    ren,col=data.shape
    r,c=clusters.shape
    return ((v1/(r-1))/(v2/(ren-r)))

```

[24]: #Métricas para 6 atributos kmeans

```

valores_unicos, frecuencias = np.unique(lab, return_counts=True)
v1=bgss(data_b.to_numpy(),res,frecuencias)
v2=wgss(data_b,lab,res)
cal=calinski(v1,v2,data_b,res)
print(cal)

```

271743655705.28818

[25]: # Métrica para 6 atributos jerárquico

```

# numero de clusters
nc=3
valores_unicos, frecuencias = np.unique(cent[-nc], return_counts=True)
t=0
for a in valores_unicos:
    cent[-nc]=np.where(cent[-nc]==a,t,cent[-nc])
    #label=np.where(label==p_min[0][0],p_min[1][0],label)
    valores_unicos=np.where(valores_unicos==a,t,valores_unicos)
    t+=1
cent_jerarquico1=calc_centroide(cent[-nc],data_b.to_numpy(),d_euc) #
v1=bgss(data_b.to_numpy(),cent_jerarquico1,frecuencias)
v2=wgss(data_b,cent[-nc],cent_jerarquico1)
cal=calinski(v1,v2,data_b,res)
print(cal)

```

[0 1 2]  
[ 2 2 396]

283646668.5482213

## 6.1 Uso de PCA

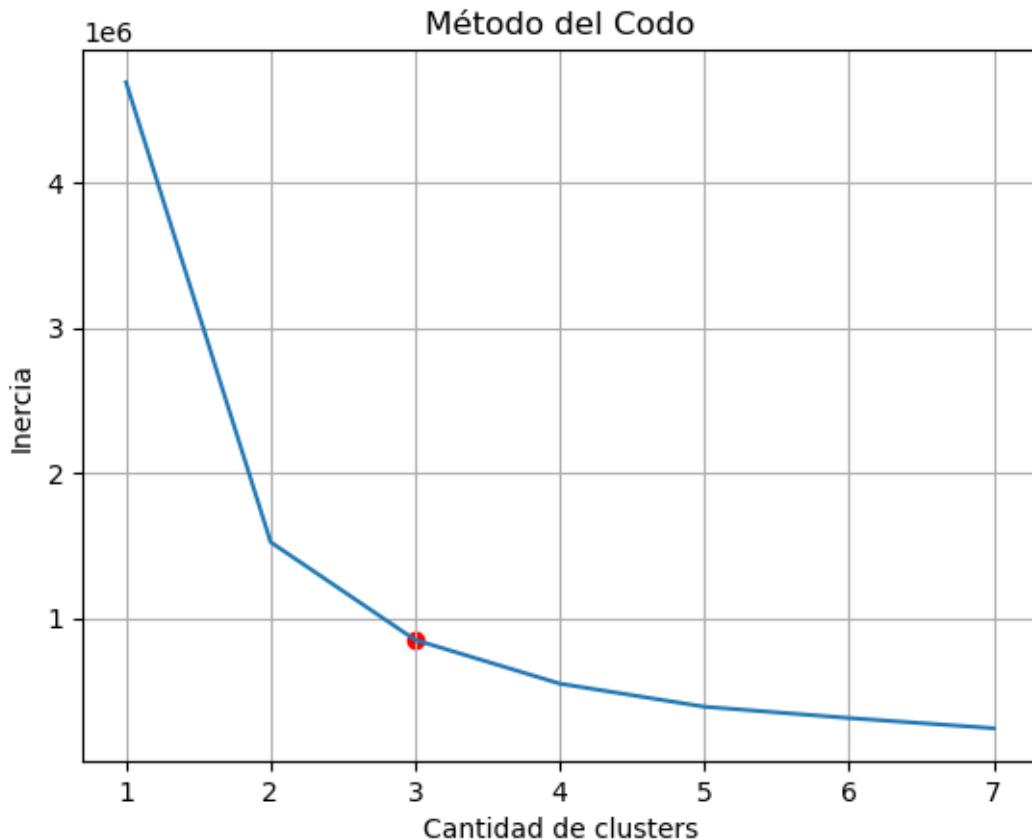
```
{'d_cg': 36.04072130342554, 'd_lab': 21.34712266379575, 'd_nas_men': 16.032413373957308, 'd_nas_sub': 12.442237832788472, 'd_gon_ment': 8.892535879862113, 'd_sup_inf': 5.244968946170824}
```

se puede considerar 4 atributos como los más importantes

```
[26]: db2=data_b[['d_cg','d_lab','d_nas_men','d_nas_sub']]
```

```
[27]: ## Kmeans  
valor=3  
inercia,n_cl=e_met(db2.to_numpy(),7,d_euc)  
plt.scatter(valor,inercia[valor-1],color='red')
```

```
[27]: <matplotlib.collections.PathCollection at 0x1f7f40d7f40>
```



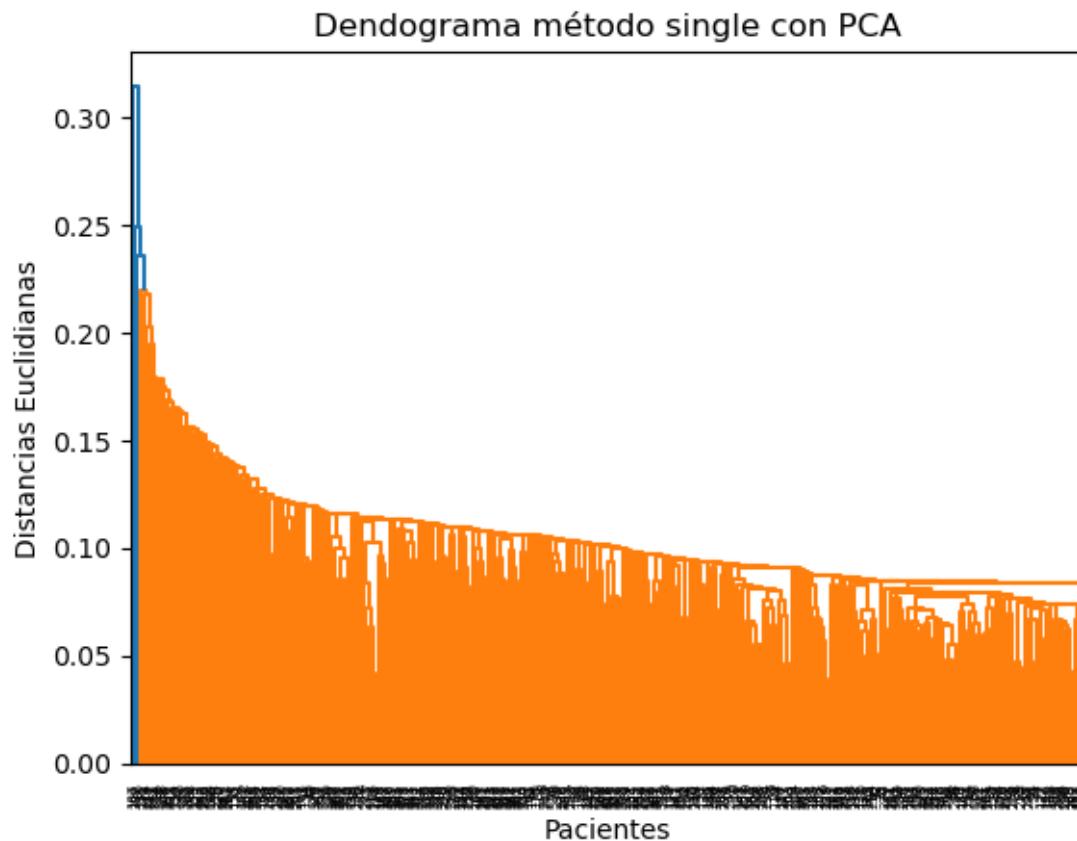
```
[28]: res,lab2=k_means(db2.to_numpy(),3,d_euc)  
res
```

```
[28]: array([[402.06225828, 211.94061219, 470.15935354, 527.47589004],  
           [380.73076954, 193.28562875, 556.23990077, 552.19316474],  
           [480.27095901, 201.70119434, 595.24238698, 595.26329982]])
```

```
[70]: # Métrica con 3 atributos  
valores_unicos, frecuencias = np.unique(lab, return_counts=True)  
v1=bgss(db2.to_numpy(),res,frecuencias)  
v2=wgss(db2,lab,res)  
cal=calinski(v1,v2,data_b,res)  
print(cal)
```

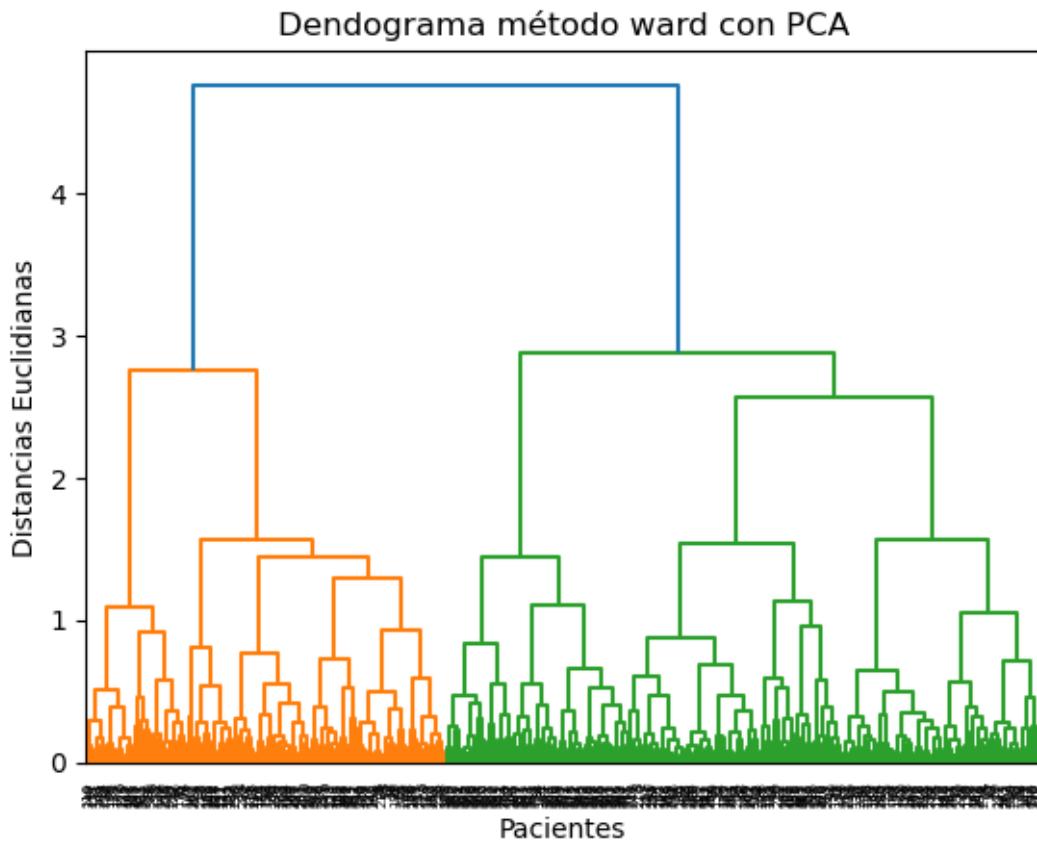
88219130996.91548

```
[67]: ## kmeans jerarquico  
import scipy.cluster.hierarchy as sch  
dendrogram = sch.dendrogram(sch.linkage(norm_min_max(db2,0,1), method ='single'))  
  
plt.title('Dendograma método single con PCA')  
plt.xlabel('Pacientes')  
plt.ylabel('Distancias Euclidianas')  
plt.show()
```



```
[69]: ## kmeans jerarquico
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(norm_min_max(db2,0,1), method = 'ward'))

plt.title('Dendograma método ward con PCA')
plt.xlabel('Pacientes')
plt.ylabel('Distancias Euclidianas')
plt.show()
```



```
[32]: import time
t1=time.time()
t2=time.time()
print('tiempo: ',t2-t1)
```

```
inicio
tiempo: 72.25643992424011
```

```
[33]: nc=3
valores_unicos, frecuencias = np.unique(cent[-nc], return_counts=True)
t=0
for a in valores_unicos:
    cent[-nc]=np.where(cent[-nc]==a,t,cent[-nc])
    #label=np.where(label==p_min[0][0],p_min[1][0],label)
    valores_unicos=np.where(valores_unicos==a,t,valores_unicos)
    t+=1
cent_jerarquico2=calc_centroide(cent[-nc],db2.to_numpy(),d_euc)
v1=bgss(db2.to_numpy(),cent_jerarquico2,frecuencias)
v2=wgss(db2,cent[-nc],cent_jerarquico2)
cal=calinski(v1,v2,data_b,res)
print(cal)
```

[0 1 2]  
[107 23 270]  
23017285642.99188

## 7 Método ward

```
[34]: import numpy as np
from scipy.cluster.hierarchy import linkage, fcluster

linkage_matrix = linkage(data_b.to_numpy(), method='ward')
```

```
[35]: threshold = 2.5
clusters = fcluster(linkage_matrix, threshold, criterion='distance')
```

```
[36]: centroids = []
for cluster_id in np.unique(clusters):
    points = data_b.to_numpy()[clusters == cluster_id]
    centroid = np.mean(points, axis=0)
    centroids.append(centroid)

# Imprimir los centroides de los clusters
#for i, centroid in enumerate(centroids):
#    print(f"Centroide del Cluster {i+1}: {centroid}")
```

```
[37]: centroides=np.asarray(centroids)
```

```
[38]: centroides.shape
```

[38]: (400, 6)

```
[39]: centroides_ward1,labels=k_means(centroides,3,d_euc)
```

```
[40]: centroides_ward1.shape
```

[40]: (3, 6)

```
[41]: lab_ward1=ajustar(centroides_ward1,data_b.to_numpy(),d_euc)
```

```
[42]: lab_ward1
#Métricas
valores_unicos, frecuencias = np.unique(lab_ward1, return_counts=True)
v1=bgss(data_b.to_numpy(),res,frecuencias)
v2=wgss(data_b,lab_ward1,centroides_ward1)
cal=calinski(v1,v2,data_b,centroides_ward1)
print(cal)
```

197819512201.3204

```
[43]: ## Con PCA
linkage_matrix = linkage(db2.to_numpy(), method='ward')
threshold = 0.01
clusters_wardpca = fcluster(linkage_matrix, threshold, criterion='distance')

centroids_ward_pca = []
for cluster_id in np.unique(clusters_wardpca):
    points = db2.to_numpy()[clusters_wardpca == cluster_id]
    centroid = np.mean(points, axis=0)
    centroids_ward_pca.append(centroid)
```

```
[44]: centroids_ward_pca=np.asarray(centroids_ward_pca)
#print(centroids_ward_pca.shape)
```

```
[45]: centroides_ward2,labels2=k_means(centroids_ward_pca,3,d_euc)
lab_ward2=ajustar(centroides_ward2,db2.to_numpy(),d_euc)
```

```
[46]: #Métricas ward pca
valores_unicos, frecuencias = np.unique(lab_ward2, return_counts=True)
v1=bgss(db2.to_numpy(),centroides_ward2,frecuencias)
v2=wgss(db2,lab_ward2,centroides_ward2)
cal=calinski(v1,v2,db2,centroides_ward2)
print(cal)
```

112108746729.70982

```
[48]: ## Calcular algun porcentaje de similitud con los otros métodos
lab_h1=ajustar(cent_jerarquico1,data_b.to_numpy(),d_euc)
lab_h2=ajustar(cent_jerarquico2,db2.to_numpy(),d_euc)
```

```
[49]: # lab, lab2, labels1, labels2
```

```
[56]: # similitud entre métodos.
met=list([lab,lab2,lab_h1,lab_h2,lab_ward1,lab_ward2])
```

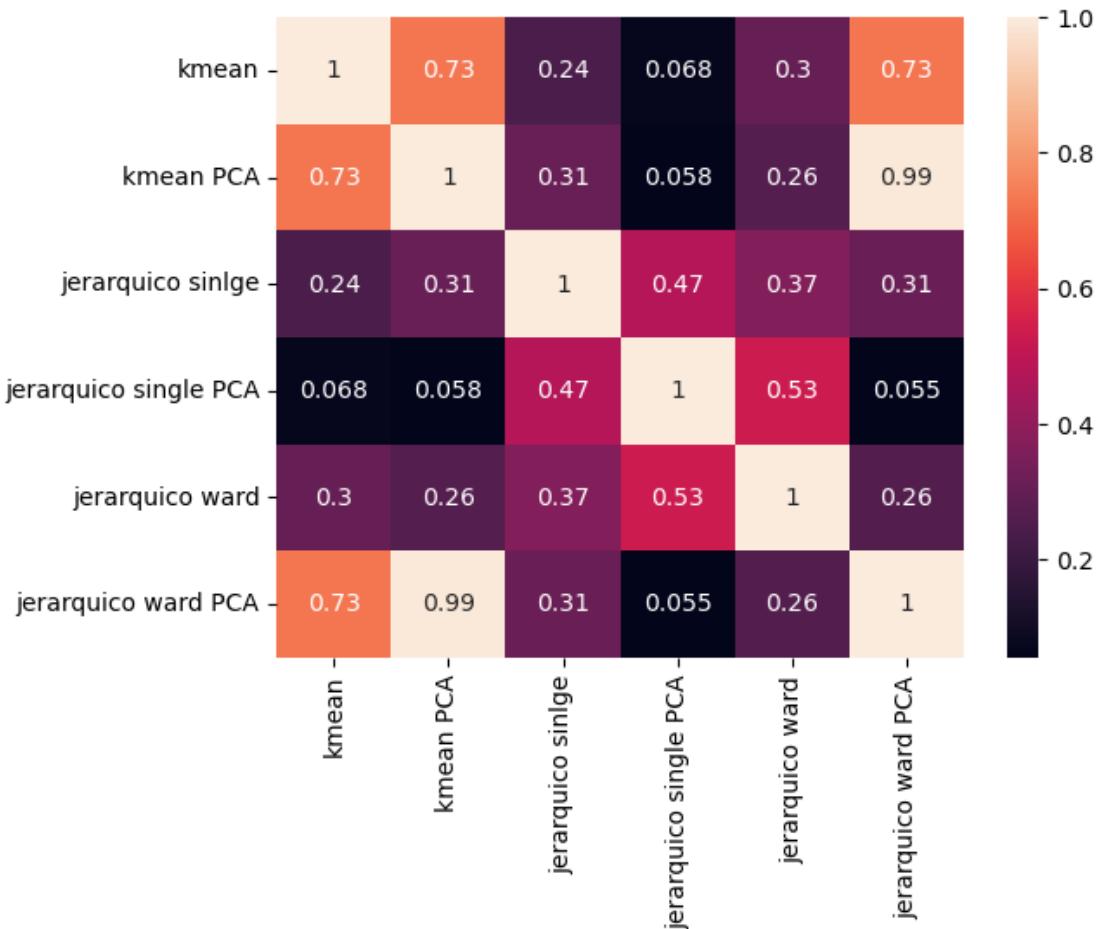
```
[58]: m_sim=np.zeros((6,6))
for a in range(len(met)):
    for b in range(len(met)):
        m_sim[a,b]=np.count_nonzero(met[a]==met[b])/400
```

```
[59]: m_sim
```

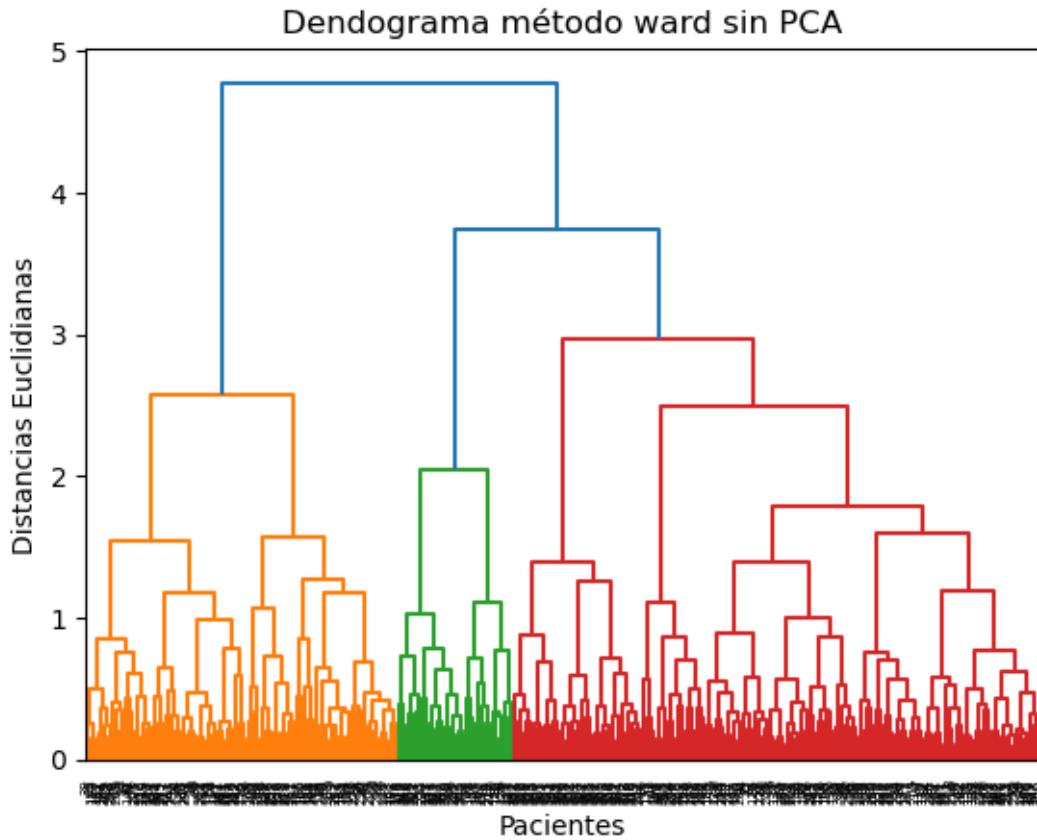
```
[59]: array([[1.      , 0.7275, 0.2425, 0.0675, 0.3025, 0.73  ],
       [0.7275, 1.      , 0.3075, 0.0575, 0.2625, 0.9925],
       [0.2425, 0.3075, 1.      , 0.475 , 0.3675, 0.31  ],
       [0.0675, 0.0575, 0.475 , 1.      , 0.53  , 0.055 ],
       [0.3025, 0.2625, 0.3675, 0.53  , 1.      , 0.26  ],
       [0.73  , 0.9925, 0.31  , 0.055 , 0.26  , 1.      ]])
```

```
[63]: xlab=['kmean','kmean PCA','jerarquico sinlge','jerarquico singlePCA',
         'jerarquico ward','jerarquico ward PCA']
sns.heatmap(m_sim,annot=True,xticklabels=xlab,yticklabels=xlab)
```

```
[63]: <AxesSubplot:>
```



```
[66]: import scipy.cluster.hierarchy as sch  
dendrogram = sch.dendrogram(sch.linkage(norm_min_max(data_b),0,1), method = 'ward')  
  
plt.title('Dendograma método ward sin PCA')  
plt.xlabel('Pacientes')  
plt.ylabel('Distancias Euclidianas')  
plt.show()
```



```
[80]: aa,bb=m_sim.shape  
summ=np.zeros((aa,1))  
for a in range(aa):  
    summ[a]=sum(m_sim[a,:])  
  
print((summ-1)/5)
```

```
[[0.414 ]  
 [0.4695]]
```

```
[0.3405]
[0.237 ]
[0.3445]
[0.4695]]
```

[76]: df.head()

```
[76]:    1_x   1_y   2_x   2_y   3_x   3_y   4_x   4_y   5_x   5_y   ...   15_x   15_y   \
0  806  1028  1384  972  1258  1218  591  1212  1406  1579  ...  1530  1546
1  847  1121  1407  1118  1250  1362  601  1218  1331  1671  ...  1417  1666
2  819  1084  1457  1105  1282  1337  563  1232  1359  1625  ...  1498  1670
3  782  1003  1369  936  1236  1200  570  1178  1373  1448  ...  1487  1420
4  799  1039  1469  955  1264  1225  623  1204  1447  1571  ...  1574  1536

      16_x   16_y   17_x   17_y   18_x   18_y   19_x   19_y
0  1471  2136  954  1447  1366  1504  649  1261
1  1388  2245  941  1533  1383  1600  741  1441
2  1467  2192  947  1530  1393  1602  604  1302
3  1429  2003  958  1401  1420  1371  613  1239
4  1644  2167  983  1445  1441  1487  662  1353
```

[5 rows x 38 columns]

[82]: d\_ceph.tail()

```
[82]:    image_path   1_x   1_y   2_x   2_y   3_x   3_y   4_x   4_y   5_x   5_y   ...   15_x   \
145  057.jpg  760  1059  1359  933  1243  1199  559  1277  1375  1464  ...  1464
146  038.jpg  815  1042  1379  1074  1250  1307  569  1211  1291  1455  ...  1455
147  013.jpg  753  1037  1348  1018  1171  1276  526  1192  1257  1383  ...  1383
148  003.jpg  761  1105  1329  911  1250  1181  622  1279  1436  1524  ...  1524
149  037.jpg  808  1025  1393  1050  1233  1299  581  1200  1318  1401  ...  1401

      15_y   16_x   16_y   17_x   17_y   18_x   18_y   19_x   19_y
145  1417  1364  1938  975  1410  1381  1382  671  1370
146  1603  1260  2094  908  1471  1331  1562  626  1318
147  1546  1356  2045  898  1425  1299  1507  603  1247
148  1369  1526  1923  1008  1404  1420  1350  712  1348
149  1620  1470  2169  922  1449  1344  1544  647  1259
```

[5 rows x 39 columns]

[88]: lab[:10, ]

```
[88]: array([[2],
       [1],
       [2],
       [1],
       [2],
```

```
[1],  
[1],  
[1],  
[0],  
[1])
```

```
[89]: d_ceph.head(10)
```

```
[89]:   image_path  1_x   1_y   2_x   2_y   3_x   3_y   4_x   4_y   5_x ... 15_x \
0    017.jpg    806  1028  1384   972  1258  1218   591  1212  1406 ... 1530
1    121.jpg    847  1121  1407  1118  1250  1362   601  1218  1331 ... 1417
2    033.jpg    819  1084  1457  1105  1282  1337   563  1232  1359 ... 1498
3    093.jpg    782  1003  1369   936  1236  1200   570  1178  1373 ... 1487
4    052.jpg    799  1039  1469   955  1264  1225   623  1204  1447 ... 1574
5    045.jpg    739  1084  1319   935  1236  1191   588  1248  1366 ... 1485
6    021.jpg    816  1089  1425  1037  1256  1308   577  1250  1269 ... 1446
7    029.jpg    765  1038  1396   970  1237  1224   647  1210  1411 ... 1512
8    080.jpg    819  1082  1354  1079  1226  1288   584  1198  1294 ... 1422
9    030.jpg    822  1056  1345  1007  1247  1267   597  1262  1313 ... 1458
```

```
      15_y   16_x   16_y   17_x   17_y   18_x   18_y   19_x   19_y
0  1546  1471  2136   954  1447  1366  1504   649  1261
1  1666  1388  2245   941  1533  1383  1600   741  1441
2  1670  1467  2192   947  1530  1393  1602   604  1302
3  1420  1429  2003   958  1401  1420  1371   613  1239
4  1536  1644  2167   983  1445  1441  1487   662  1353
5  1455  1384  1961   946  1415  1373  1409   639  1298
6  1634  1274  2221   901  1508  1335  1552   643  1367
7  1488  1440  2074   973  1452  1406  1434   660  1363
8  1522  1302  2060   921  1403  1333  1481   646  1285
9  1575  1390  2134   946  1501  1315  1523   674  1351
```

```
[10 rows x 39 columns]
```

```
[ ]:
```