Machine learning y Deep learning con Python: Lectura 7

Ing. Pedro Rotta

Universidad de Piura - Vida Universitaria

Enero-2022

Métodos no supervizados

Estos métodos no tienen una forma de medir su precisión, ya que no usan etiquetas, por lo que uno de los retos en este tipo de aprendizaje, es cómo medir su eficiencia.

Métodos no supervizados

Estos métodos no tienen una forma de medir su precisión, ya que no usan etiquetas, por lo que uno de los retos en este tipo de aprendizaje, es cómo medir su eficiencia.

Es por esta razón que el uso de los métodos no supervizados radica más en dos casos de uso:

Métodos no supervizados

Estos métodos no tienen una forma de medir su precisión, ya que no usan etiquetas, por lo que uno de los retos en este tipo de aprendizaje, es cómo medir su eficiencia.

Es por esta razón que el uso de los métodos no supervizados radica más en dos casos de uso:

Para comprimir dimensionalmente la data, mediante **métodos de reducción dimensional**, así podemos explotar mejor nuestro conjunto de datos antes de realizar un aprendizaje supervizado y visualizarlo de una mejor manera.

Métodos no supervizados

Estos métodos no tienen una forma de medir su precisión, ya que no usan etiquetas, por lo que uno de los retos en este tipo de aprendizaje, es cómo medir su eficiencia.

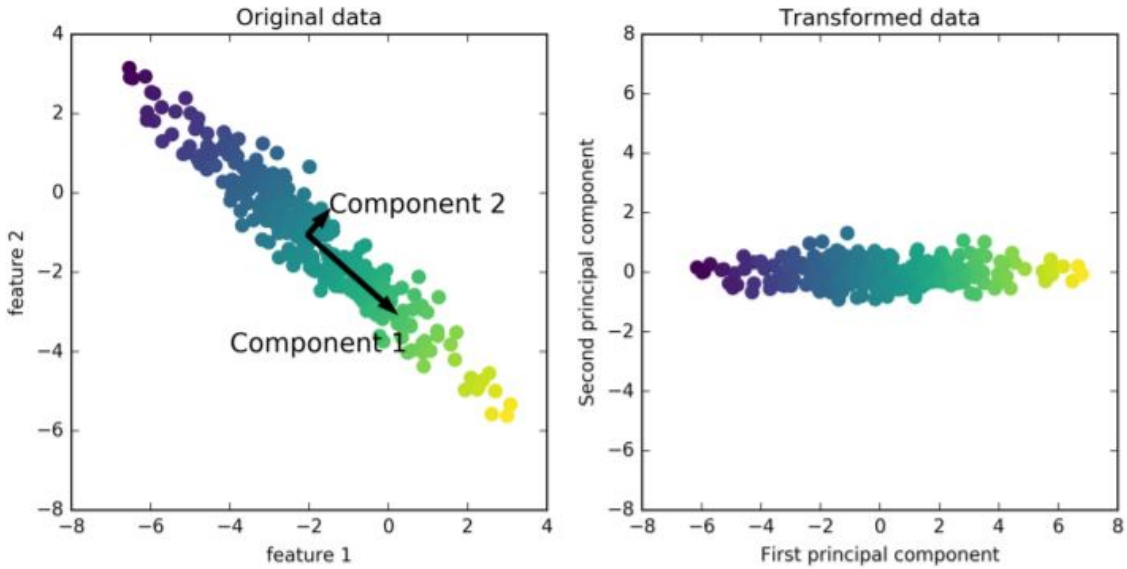
Es por esta razón que el uso de los métodos no supervizados radica más en dos casos de uso:

Para comprimir dimensionalmente la data, mediante **métodos de reducción dimensional**, así podemos explotar mejor nuestro conjunto de datos antes de realizar un aprendizaje supervizado y visualizarlo de una mejor manera.

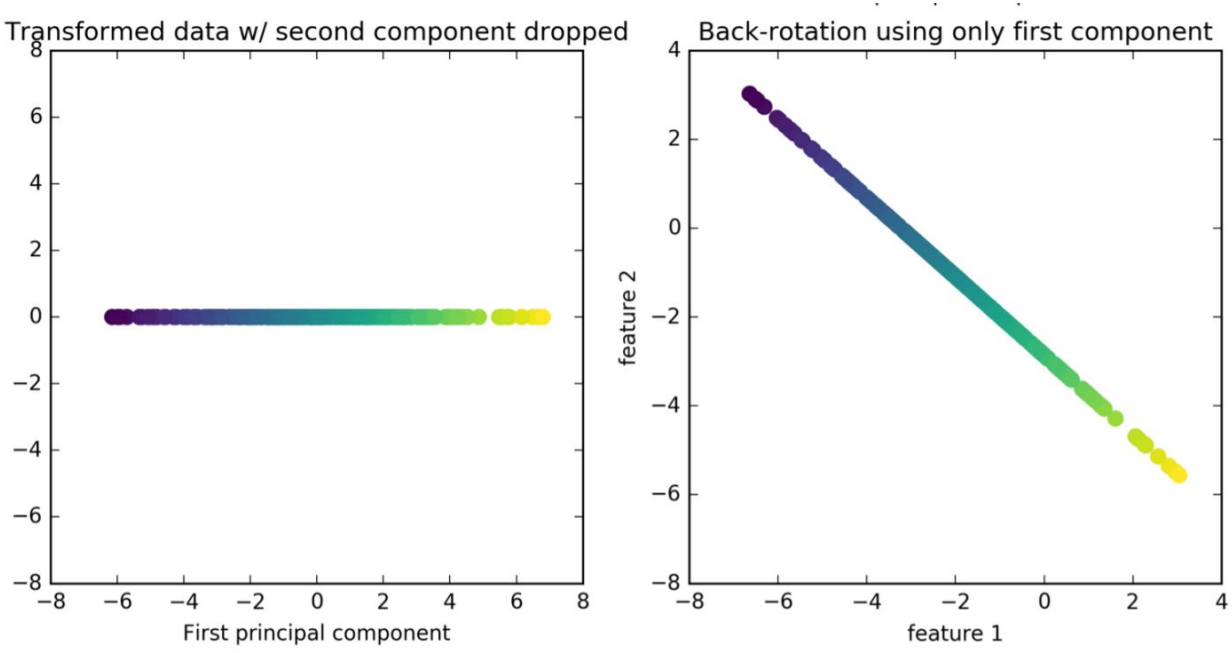
El otro campo de uso es el **agrupamiento de datos por características comunes** aka **clustering** de datos. Este tipo de algoritmos permite agrupar los datos por sus características similares, esto es útil cuando no se tiene un conjunto de datos balanceado, o cuando no se tienen las etiquetas a las que pertenecen los datos y luego aplicar un aprendizaje supervizado.

Análisis de componentes principales

El análisis de componentes principales aka **PCA** (Principal Component Analysis) es un método por el cual se puede reducir la dimensionalidad de la data dadas variables correlacionadas. En 2 dimensiones, lo que hace el método es rotar la data en los ejes principales y eliminar la media de los datos para que la media se convierta en 0. Así se puede reducir la dimensionalidad.

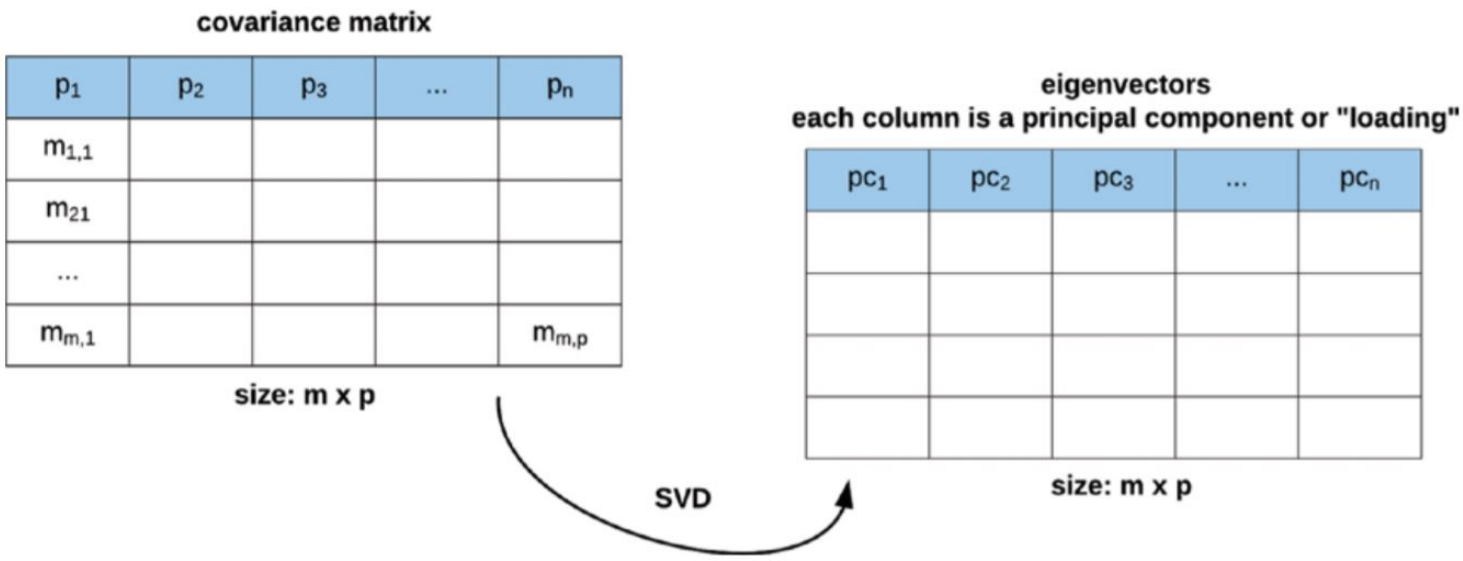


Análisis de componentes principales



Análisis de componentes principales

Para más de 2 dimensiones, el método realiza la matriz de covarianza del conjunto de datos. La matriz de covarianza captura la relación linear entre las características. Luego de ello, el algoritmo encuentra los vectores propios usando el método valores de descomposición singular, aka **SVD**.



Análisis de componentes principales

La matriz de componentes principales resultantes es una matriz cuadrada *pxp* que expresa en cada columna a los vectores propios de cada característica original.

Análisis de componentes principales

La matriz de componentes principales resultantes es una matriz cuadrada *pxp* que expresa en cada columna a los vectores propios de cada característica original.

Para reducir la dimensionalidad usando esta matriz de valores singulares, lo que se necesita hacer es realizar una multiplicación matricial interna entre la matriz del conjunto de datos y los componentes principales que se desean obtener. Por ejemplo, en el caso se desee una reducción dimensional a n $= 2$

Análisis de componentes principales

La matriz de componentes principales resultantes es una matriz cuadrada *pxp* que expresa en cada columna a los vectores propios de cada característica original.

Para reducir la dimensionalidad usando esta matriz de valores singulares, lo que se necesita hacer es realizar una multiplicación matricial interna entre la matriz del conjunto de datos y los componentes principales que se desean obtener. Por ejemplo, en el caso se desee una reducción dimensional a n $= 2$

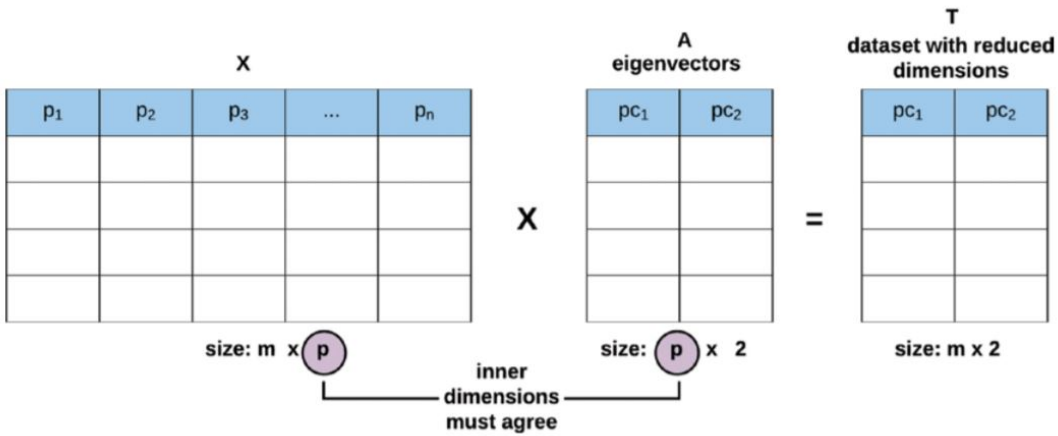
$$T_{reduced} = X_{mxp}.A_{px2}$$

Análisis de componentes principales

La matriz de componentes principales resultantes es una matriz cuadrada *pxp* que expresa en cada columna a los vectores propios de cada característica original.

Para reducir la dimensionalidad usando esta matriz de valores singulares, lo que se necesita hacer es realizar una multiplicación matricial interna entre la matriz del conjunto de datos y los componentes principales que se desean obtener. Por ejemplo, en el caso se desee una reducción dimensional a n = 2

$$T_{reduced} = X_{mxp}.A_{px2}$$



Computarización

Para computarizar PCA usamos del módulo **decomposition**:

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Pero ajustar el modelo y realizar la transformación, la sintaxis será:

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Pero ajustar el modelo y realizar la transformación, la sintaxis será:

```
pca.fit(X_data)
```

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Pero ajustar el modelo y realizar la transformación, la sintaxis será:

```
pca.fit(X_data)
```

Para calcular los X principales, es decir, con reducción de dimensión. La sintaxis será:

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Pero ajustar el modelo y realizar la transformación, la sintaxis será:

```
pca.fit(X_data)
```

Para calcular los X principales, es decir, con reducción de dimensión. La sintaxis será:

```
X_pca = pca.transform(X_data))
```

Computarización

Para computarizar PCA usamos del módulo **decomposition**:

```
from sklearn.decomposition import PCA
```

Cuando se crea el objeto PCA, se van a crear los componentes principales, pero de dimensionalidad igual a los features originales. Para reducir los features, tenemos que colocarlo como propiedad del objeto.

```
pca = PCA(n_components=2)
```

Pero ajustar el modelo y realizar la transformación, la sintaxis será:

```
pca.fit(X_data)
```

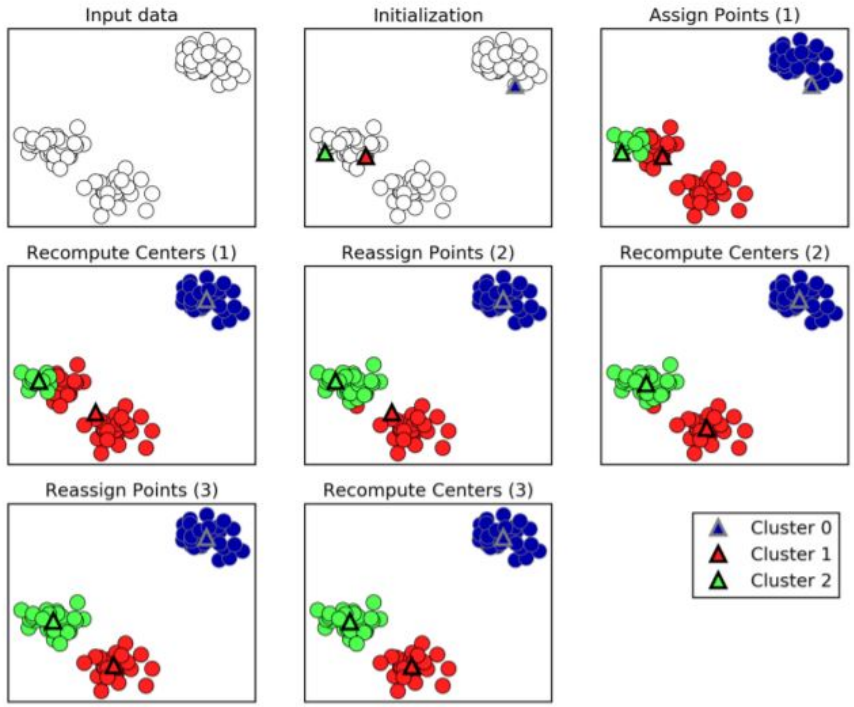
Para calcular los X principales, es decir, con reducción de dimensión. La sintaxis será:

```
X_pca = pca.transform(X_data))
```

Ver problema 1

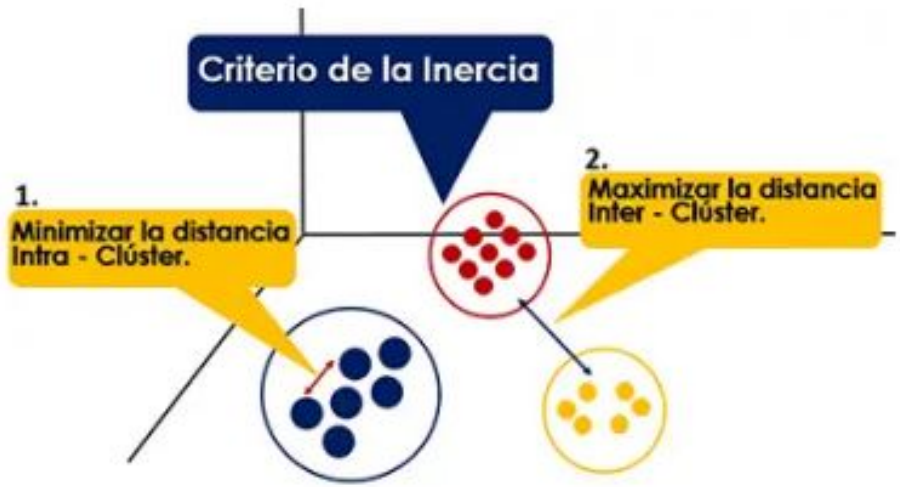
K-means clustering

Es otra técnica de aprendizaje no supervisado. En este caso, se trata de agrupar conjuntos de datos, según sus propiedades particulares. Lo que hace el algoritmo, es dada un número K de clases por clasificar, inicializa el valor de los centroides de cada clase, y luego en cada iteración calcula la distancia de los objetos al centroide, cambiando de posición el centroide.



K means clustering

Para poder definir qué tan bien se separan las clases se define un criterio denominado, **criterio de inercia**, que busca minimizar la distancia entre los puntos dentro de cada grupo y que los grupos generados, estén distanciados.



K means clustering

Se definen 3 tipos de inercia: La inercia intraclases, que es el promedio de la suma de los datos dentro de las clases y el centro de gravedad al que pertenecen, expresado como:

$$wcss = \frac{1}{n} \sum_{k=1}^{k} \sum_{i \epsilon c_k} ||x_i - g_k||^2$$

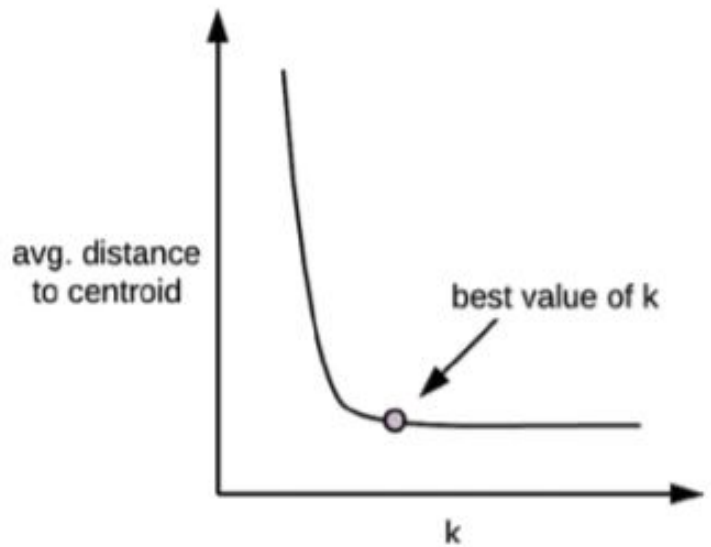
La inercia interclases, que es el promedio de las distancias que hay entre los centros de gravedad y se expresa como:

$$bpp = \sum_{k=1}^{k} \frac{|c_k|}{n} ||g_k - g||^2$$

Y la inercia total, que es la suma de ambas.

K means clustering

Algo importante de definir para el método K-means es el número de clases K, que a veces no está claro, sobre todo para un análisis multidimensional. Por lo que se recurre al método del codo de Jambú, que grafica la relación entre el número de clusters vs la inercia intraclases.



Computarización Kmeans

Para computarizar el K-means, se importa del módulo cluster.

```
from sklearn.cluster import Kmeans
```

Para hallar el mejor K, primero usamos el método del codo de jambu, para ello realizamos la siguiente computarización:

```
wcss = []
for i in range(1,11):
   kmeans = KMeans(n_cluster = i, max_iter = 300)
   kmeans.fit(X_scalado))
   wcss.append(kmeans.inertia_)
   plt.plot(range(1,11),wcss)
```

Computarización Kmeans

Una vez encontrado el K óptimo, podemos computarizar el separador mmediante **KMeans**. Luego se puede obtener el valor de las etiquetas usando el método _labels

```
labels = modelclustering.labels_
```

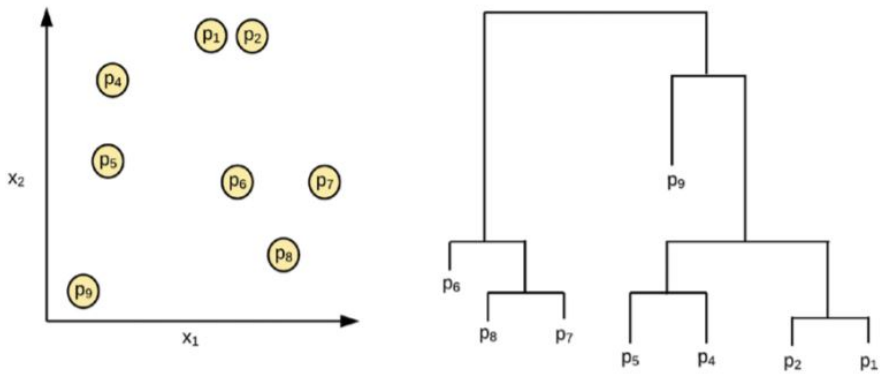
Para realizar un gráfico podemos usar PCA primero para poder hallar 2 componentes principales para reducir los datos y luego verificar la separación de los datos producido por el método K-means.

Ver problema 2

Agrupación Jerárquica

La agrupación jerárquica es otro método de separación por clases que a diferencia del método k-means no necesita una cantidad de clases K, sino que usa un método de separación por clases aglomerativo o divisivo.

En este tipo de métodos, se hace una separación escalable que da como resultado un dendograma.



Agrupación Jerárquica

El método de clases aglomerativo toma como principio que todos los puntos, en el conjunto de datos son clases distintas, luego calcula la distancia entre puntos y realiza una combinación de los puntos más cercanos y calcula su centroide. Luego en la siguiente iteración, calcula la distancia entre los centroides más cercanos de los grupos formados, formando nuevos grupos, y así sucesivamente hasta formar un solo grupo.

En el método divisivo, es al reves, en este método, todo se genera como un grupo grande, y luego se calcula distancias entre los puntos y se elije a los más separados como 2 grupos distintos, donde estos pasan a ser centroides, y así sucesivamente hasta que quedan todos los conjuntos de datos como clases.

Agrupación Jerárquica

Para saber que tan similares o desiguales son los diferentes grupos se usa la vinculación o **linkage**. Existen varias formas de calcular la vinculación, las más conocidas son: centroide, promedio , completa, unitaria y manhattan.

Dependiendo del tipo de vinculación escogido, el dendograma tendrá diferentes tipos de corte, y por lo tanto se formarán diferentes formas de agrupar la data

Computarización

Para poder usar la agrupación jerárquica se tiene que importar de scipy mediante:

```
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import
dendogram,linkage,fcluster
X = np.array(data)
```

Para crear el clustering jerárquico escribimos:

```
Clust_jera = linkage(X,method = 'complete')
```

Para crear el dendograma:

```
dendogram = sch.dendogram(Clust_jera)
```

Para generar las clases a un determinado corte según el dendograma:

```
cluster = fcluster(Clust_jera, t=2,
criterion= 'distance')
```

Metodología

Cómo último, podemos plasmar una metodología para los métodos de Machine learning vistos en el curso: