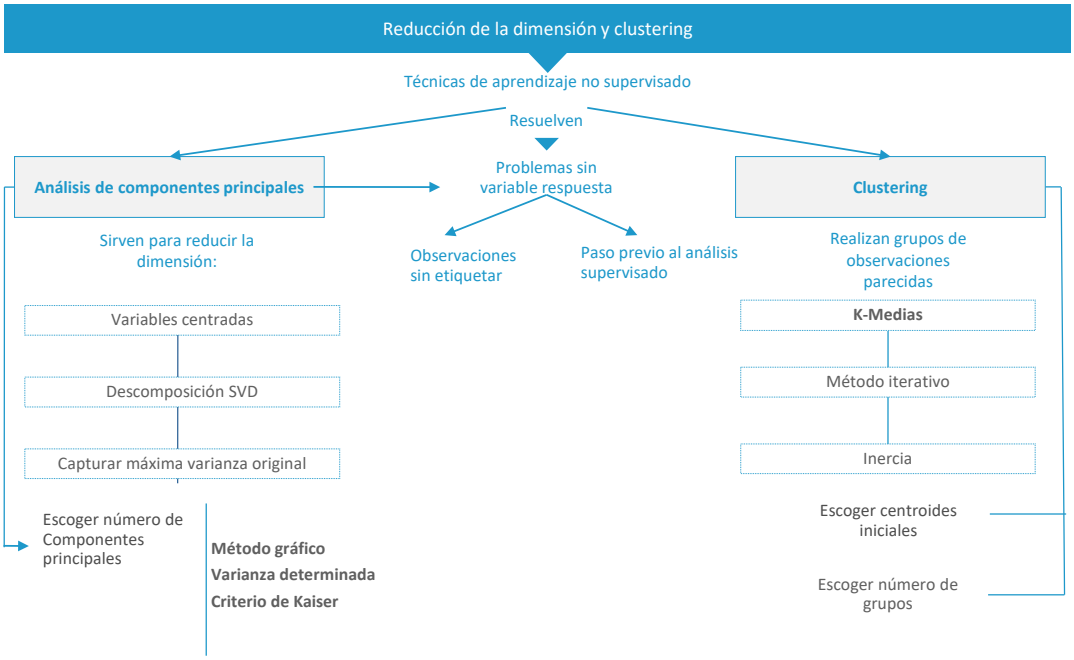
Técnicas Multivariantes

Reducción de la dimensión y clustering

Índice

Esquema



Ideas clave

9.1 Introducción y objetivos

En este tema se estudiarán los métodos de reducción de la dimensión e introduciremos los métodos de aprendizaje no supervisado, centrándonos fundamentalmente en el *clustering*.

En algunas aplicaciones de Machine Learning nos podemos encontrar con cientos, miles e incluso con millones de variables predictoras para cada observación de la muestra de entrenamiento. Esto, además de hacer que encontrar una solución sea extremadamente costoso computacionalmente, también hace que sea muy difícil encontrar una buena solución. Afortunadamente, en muchas ocasiones, es posible reducir considerablemente el número de variables del problema. Uno de los métodos más destacados para lograr reducir la dimensionalidad de un conjunto de datos es el análisis de las componentes principales, que es un método basado en la proyección de los datos desde el sistema de referencia que fijan las coordenadas originales a otro sistema con un número de dimensiones menor. Posteriormente, se detallarán algunos de los métodos de aprendizaje no supervisado donde se abordarán los distintos tipos de clustering. El clustering consiste en encontrar patrones similares en distintas observaciones para poder formar grupos.

9.2 Análisis de componentes principales

El análisis de componentes principales (PCA, por sus siglas en inglés) es uno de los métodos más empleados para tratar de reducir la dimensión de un conjunto de datos.

En este método, se trata de trasladar los datos desde el sistema de referencia con n dimensiones original a otro sistema de referencia con un número de dimensiones menor a n. Para lograr este objetivo, primero se identifica un hiperplano próximo cercano a los datos y después se proyectan dichos datos sobre él, tal y como se muestra en la Figura 1.

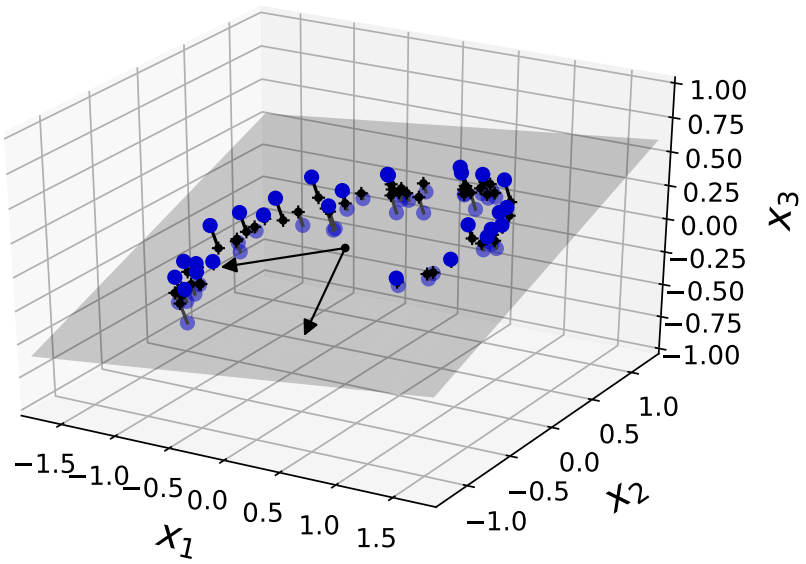


Figura 1: Conjunto de datos en tres dimensiones e hiperplano de proyección. Figura adaptada de (Geron, 2019)

Para escoger el hiperplano sobre el cual se van a proyectar los datos, la estrategia adecuada es escoger aquel que pueda mantener la máxima varianza posible que ofrecían los datos representados en el sistema de referencia original. Por ejemplo, en un conjunto de datos que originalmente poseen dos dimensiones, se puede realizar la proyección de dichos datos sobre un eje y reducir de dos a una dimensión. Para realizar esta proyección se pueden escoger distintos ejes. En la Figura 2 se muestra esta operación sobre un conjunto de datos de dos dimensiones y cómo la proyección varía según el eje seleccionado: la línea sólida representa al eje que mantiene la máxima varianza posible mientras que el eje de la línea punteada contiene la mínima varianza posible (el eje de la línea discontinua recoge una varianza intermedia). Al escoger la proyección que recoja la máxima cantidad de varianza posible estaremos perdiendo la

menor cantidad de información que proporcionan los datos. Además, esta proyección coincide con la mínima distancia al cuadrado entre los datos originales y todas las posibles proyecciones.

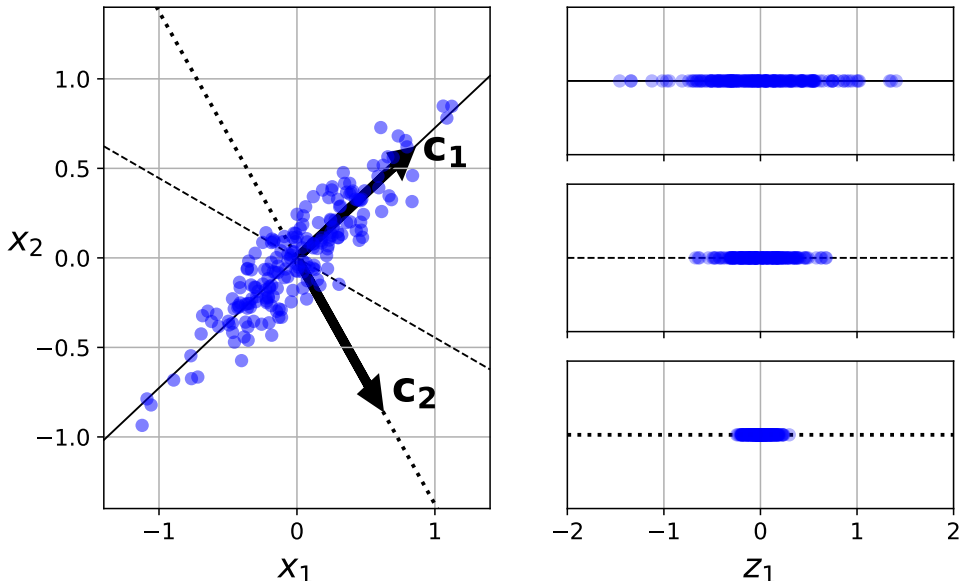


Figura 2: Conjunto de datos en dos dimensiones y distintos ejes de proyección. Figura adaptada de (Geron, 2019)

De este modo, el método de componentes principales identifica el eje que consigue capturar la mayor cantidad de varianza en el conjunto de datos original (en la Figura 2 la línea sólida). Después, identifica un segundo eje, ortogonal al primero, que recoge la máxima varianza restante posible. Esto se repite hasta el número de dimensiones del problema. En el ejemplo de la figura 2 se obtendría hasta un segundo eje (la línea punteada). En este método, el primer eje es la primera componente principal, el segundo eje es la segunda componente principal y así sucesivamente.

Para encontrar estas componentes principales se emplea la técnica de descomposición matricial (SVD), donde V contiene los vectores unitarios que definen cada una de las componentes principales. Para obtener las componentes principales en *python* podemos hacer uso de esta descomposición mediante la librería *numpy*. Antes de realizar la descomposición con *numpy* es necesario centrar cada una de las variables antes de aplicar la técnica de descomposición matricial SVD. Como veremos más adelante, en

Scikit-Learn, la clase que busca las componentes principales ya implementa de forma automática dicho centrado. En el trozo de código que se muestra a continuación se muestra como realizar esta descomposición empleando *numpy*.

```
# importar numpy-----------------------------------------------
import numpy as np
# descomp matricial SVD----------------------------------------
# centrar vector variables
X_centered = X - X.mean(axis=0)
# aplicar descomp SVD
U, s, Vt = np.linalg.svd(X_centered)
# obtener primeras 2 componentes
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

Una vez que se han identificado las componentes principales, se puede reducir la dimensionalidad del conjunto de datos n a un número menor de dimensiones *d* proyectando al hiperplano que conforman las *d* primeras componentes principales. Seleccionar este hiperplano nos permite recoger la máxima varianza posible de los datos originales en menos dimensiones. Por ejemplo, en la Figura 3 se muestra la proyección resultante en el hiperplano de las dos primeras componentes principales del conjunto de datos de la Figura 1.

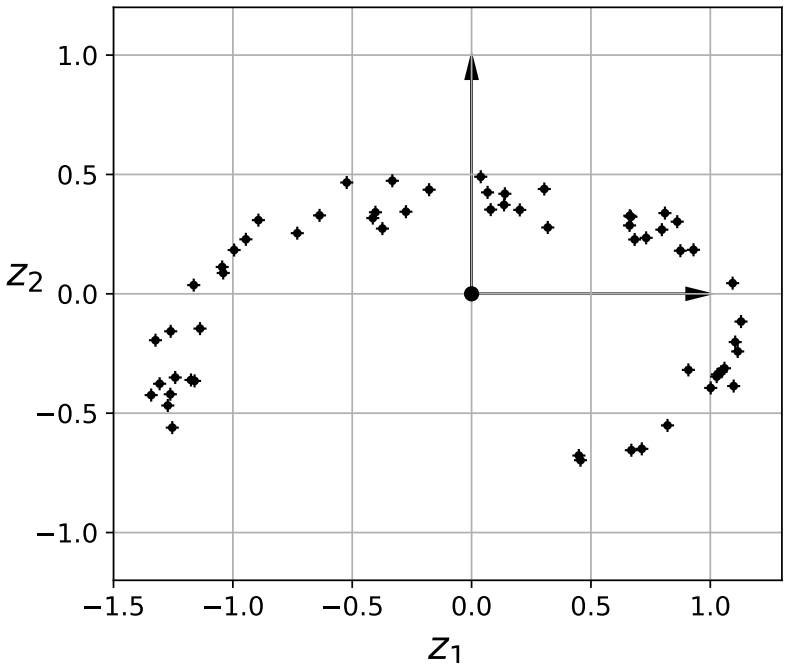


Figura 3: Conjunto de datos en dos dimensiones de la Figura 1 tras su proyección. Figura adaptada de (Geron, 2019)

Para obtener la proyección de los datos originales en el hiperplano que conforman las *d* componentes principales es necesario obtener el producto matricial de la matriz de entrenamiento X por la matriz W_d, definida como la matriz que contiene las primeras *d* columnas de V. Esta operación se puede realizar con el siguiente código:

```
# proyectar el conjunto original en 2 dimensiones
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

No obstante, no es necesario realizar todas estas operaciones con la librería *numpy* para obtener el análisis en componentes principales de un conjunto de datos. En Scikit-Learn tenemos disponible la clase PCA que emplea la descomposición *SVD* para obtener las PCA del mismo modo que hemos realizado utilizando *numpy*. Con el siguiente código se obtendría el mismor resultado para obtener la componentes principales de un conjunto de datos X.

```
# PCA con sklearn-----------------------------------------------
from sklearn.decomposition import PCA
# crear clase PCA con 2 componentes
pca = PCA(n_components = 2)
# obtener la proyeccion de la matriz X
X2D = pca.fit_transform(X)
```

Una vez que se ha empleado el método *fit_transform()*, el objeto **pca** contiene el atributo *components_* donde está la información sobre la matriz transpuesta de W_d, es decir, de los vectores unitarios que definen las componentes principales.

Otra información relevante es la proporción de varianza explicada por las *d* componentes principales. Esta información la podemos encontrar recogida en la variable *pca.explained_variance_ratio_*. En el ejemplo de las Figuras 1 y 3 se puede comprobar que las dos primeras componentes principales recogen el 98.8 % de la varianza total, por lo que parece una transformación razonable reducir la dimensión de 3 a 2.

En el caso anterior, parece obvio que 2 era el número de componentes principales a escoger adecuado, pero en otras ocasiones puede que no esté tan claro. Para escoger el número de dimensiones adecuado existen numerosas estrategias, que dependerán de cuáles son los objetivos de aplicar el análisis de componentes principales. Algunas de las estrategias más empleadas son:

- Para poder visualizar un conjunto de datos: escogeremos 2 ó 3 componentes principales.
- Para reducir el número de dimensiones capturando una varianza alta en la proyección: escoger el número de dimensiones que garantice un cierto umbral de varianza recogida (por ejemplo, del 95 %).
- Otro método para capturar gran parte de la varianza: pintar la varianza recogida por cada componente principal y buscar gráficamente dónde se produce el codo.

- Criterio de Kaiser. Emplea como criterio escoger aquellas componentes principales cuya varianza (el cuadrado de los valores de la matriz s de la descomposición SVD) sea mayor que 1.

9.3 Otros métodos de reducción de la dimensión

Además del método de componentes principales existen diferentes alternativas que también permiten reducir la dimensionalidad. Algunas de ellas son:

- Proyecciones aleatorias:

Como indica su nombre, los datos se proyectan linealmente a un espacio con una dimensión menor de manera aleatoria. En la documentación de *sklearn.random_projection* que encontrarás en la sección de **a fondo** puedes descubrir más información sobre el método.

- Isomapas:

Este método consiste en crear un grafo conectando cada observación de la muestra con sus vecinos más cercanos, para después reducir la dimensionalidad pero almacenando la información sobre la distancia geodésica (el número de nodos en el grafo) entre las observaciones.

- Análisis lineal discriminante (LDA, por sus siglas en inglés):

En realidad, es un algoritmo de clasificación, pero de modo que determina los ejes que mejor discriminan las clases, dichos ejes pueden ser empleados para definir los hiperplanos sobre los cuales realizar las proyecciones de los conjuntos de datos.

9.4 Técnicas de aprendizaje no supervisado.

Aunque la mayoría de aplicaciones del aprendizaje automático se hayan desarrollado para el aprendizaje supervisado, una gran cantidad de datos viene sin etiquetar, y por tanto, el aprendizaje no supervisado tiene un gran interés.

Veamos la utilidad del aprendizaje no supervisado con el siguiente ejemplo: queremos realizar una aplicación que determine si una pieza (o producto) de una cadena de montaje es defectuosa o no realizando una foto una vez se ha terminado. Si hacemos esto para cada pieza terminada, en poco tiempo tendremos una gran base de datos. Ahora bien, estas fotos no están etiquetadas. Es decir, no sabemos si las piezas son defectuosas o no. Una opción sería revisar todas las fotos por un experto, y que etiquete si las piezas son defectuosas o no, pero esto puede ser tedioso, lento y no exento de errores. La alternativa, sería acudir al aprendizaje no supervisado.

Entre las distintas técnicas de aprendizaje no supervisado podemos encontrar:

- **Clustering**. La meta es agrupar observaciones similares en *clusters*. El *Clustering* es una herramienta muy útil para: el análisis de datos, la segmentación de clientes, motores de búsqueda, segmentación de imágenes, reducción de la dimensionalidad, etc.

- **Detección de anomalías**. El objetivo es aprender cómo son los datos “normales”, para posteriormente detectar qué observaciones no son “normales”. Se emplea para realizar controles de calidad en líneas de producción, para encontrar anomalías en series temporales, para analizar registros, etc.

- **Estimación de densidad**. El objetivo es estimar la función de densidad de probabilidad (PDF, ver Tema 1) que subyace en la generación de los datos obtenidos. Se puede emplear para la detección de anomalías: las observaciones que se encuentren en regiones con una probabilidad muy baja pueden ser anomalías. También es muy

útil para el análisis de los datos y su visualización.

9.5 Clustering

El clustering consiste en realizar grupos de observaciones que comparten características similares. No es necesario desentrañar cómo son dichas observaciones, sino, poder agrupar a las que se parezcan entre sí. De modo similar a los problemas de clasificación, a cada observación se le asigna un grupo. Sin embargo, a diferencia de la clasificación, la asignación a los distintos grupos se realizan de forma no supervisada. En la Figura 4 se muestra para el conjunto de datos *Iris* cómo se diferencia el problema de clasificación del de *clustering*. A la izquierda de la Figura 4 se encuentran las observaciones, donde su clase viene representada con un marcador, es decir están etiquetadas. Ante este tipo de problemas se pueden aplicar técnicas de clasificación como la regresión logística, KNN, o clasificadores basados en árboles como hemos visto en temas anteriores. Por otro lado, a la derecha de la Figura 4 se muestran las mismas observaciones, pero sin etiquetar. Así pues, en este caso no se puede aplicar un método de clasificación, pero sí de clustering. El método de clustering, debe determinar cuántos grupos diferentes existen y después asignar a cada observación a qué grupo pertenece. Como veremos más adelante, la elección del número de grupos no es trivial. De hecho, con las variables que se representan en la Figura 4 se podría pensar que existen 2 grupos, en vez de los 3 grupos que realmente existen.

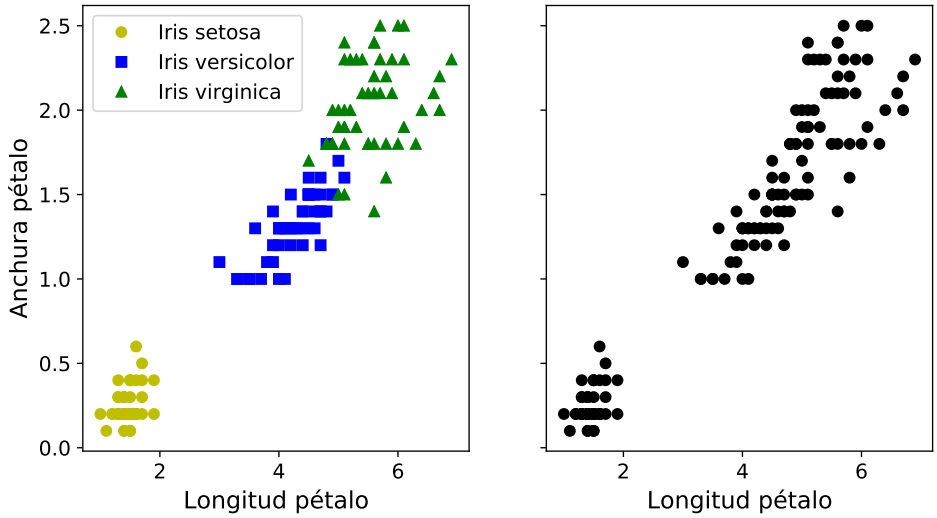


Figura 4: Clasificación (izquierda) vs Clustering (derecha)

Aplicaciones del clustering

El clustering se puede emplear para las siguientes aplicaciones:

- **Segmentación de clientes**. Se puede dividir en grupos de clientes a aquellos que tienen un comportamiento similar en cuanto a compras o en cuanto a la navegación dentro de la web. Esta aplicación es muy útil para entender quiénes son nuestros clientes y cuáles son sus preferencias y necesidades, para así poder adaptar las campañas de marketing a cada segmento.

- **Análisis de datos**. Cuando se analiza un nuevo conjunto de datos, puede ser de gran ayuda aplicar algún algoritmo de clustering, y luego analizar cada *cluster* por separado.

- **Reducción de la dimensionalidad**. Se puede conseguir reducir la dimensión de un conjunto de datos mediante el *clustering*. Primero, se realiza una división de las observaciones del conjunto de datos en *clusters* y después se obtiene la afinidad, que es una medida que indica cuánto encaja una observación en un determinado *cluster*,

para cada observación con cada uno de los diferentes *clusters*. La reducción de la dimensión se consigue transformando las variables originales de las observaciones por el vector de sus valores de afinidad. Entonces, si se han escogido *K clusters*, el nuevo número de dimensión es *K*, que suele ser mucho menor que el número de variables originales.

- **Detección de anomalías**. Las observaciones que tengan poca afinidad para todos los *clusteres* es probable que sean anomalías. Por ejemplo, si hemos realizado una segmentación por *clusters* para determinar el comportamiento de nuestros clientes a través de la página web, se pueden detectar comportamientos anómalos, como un número inusual de peticiones (clicks) por segundo. La detección de anomalías es muy útil para detectar defectos en las cadenas de montaje y producción, y para la detección de fraudes.

- **Para el aprendizaje semi-supervisado**. Si únicamente se tienen unas pocas etiquetas de las observaciones en nuestro conjunto de datos, se puede realizar una clasificación en *clusters* y después propagar las etiquetas para todas las observaciones que se hayan agrupado en el mismo *cluster*.

- **Motores de búsqueda**. Algunos motores de búsqueda, sobre todo en las aplicaciones de telefonía móvil, permiten buscar imágenes similares a partir de una imagen de referencia. Para poder construir este tipo de algoritmos es necesario dividir en *clusters* las imágenes de la base de datos (pueden ser imágenes tomadas por nosotros mismos). Cuando se proporciona la imagen de referencia, el algoritmo debe clasificar esa imagen en uno de los *clusters*, y devuelve todas las imágenes pertenecientes al mismo.

- **Segmentación de imágenes**. Otra de las aplicaciones en las que se emplea el *clustering* es para la segmentación de imágenes. Agrupando los píxeles en *clusters* según su color y luego reemplazando para cada píxel su color por el de la media del *cluster* se consigue reducir considerablemente el número de colores distintos de una imagen. Esta técnica se emplea para la detección de objetos o para sistemas de

rastreo.

9.6 Métodos de creación de clusters: K-medias

No existe una definición de lo que es un cluster. Depende del tipo de problemas y de la técnica empleada. Por ejemplo, entre otras, podemos encontrar las siguientes estrategias para crear los clusters:

- A partir de **centroides**. Algunos algoritmos buscan un punto en particular, denominado centroide, sobre el cual se distribuyen las distintas observaciones del cluster.

- Por **regiones**. Otras técnicas buscan regiones continuas de observaciones muy cercanas entre sí, pudiéndose obtener cualquier tipo de forma.

- **Jerárquicos**. También existen técnicas que generan los clusters de forma jerárquica, obteniéndose clusters dentro de clusters.

A continuación se detalla una de las técnicas más empleadas en la creación de *clusters*, la técnica de K-medias.

La técnica de K-medias (K-means) es un algoritmo sencillo capaz de crear *K* clusters a partir de las distancias de las observaciones a los centroides de cada uno de los *clusters* siguiendo el siguiente proceso iterativo:

- 1. Se generan *K* centroides, introducidos manualmente o de manera aleatoria.

- 2. Se le asigna a cada observación del conjunto de datos el *cluster* del centroide *K* más cercano.

- 3. Se computan los nuevos centroides para cada cluster *K*.

4. Se repiten los pasos 2 y 3 (se reasignan las observaciones a los nuevos clusters formados por los nuevos centroides *K* y se actulizan dichos centroides) hasta que las observaciones ya no varían más de *cluster*.

A continuación se muestra un ejemplo de aplicación del método de K-medias. Se tienen 5 grupos distintos (tres, más a la izquierda, con poca dispersión desde su centro; y dos, con una dispersión más grande en el centro y en la derecha de la figura) tal y como se observa en la Figura 5. Aplicando el método de K-medias, en pocas iteraciones, se llega a la solución que se muestra en la Figura 6. El método K-medias siempre llega a una solución (no se queda iterando infinitamente), pero puede que en algunas ocasiones, se alcance un óptimo local, tal y como se muestra en la Figura 7. Que esto ocurra depende de cómo se han seleccionado los centroides, y por lo tanto, es recomendable escoger una estrategia correcta a la hora de inicializarlos. En el ejemplo anterior, tanto para obtener la solución de la Figura 6 como para obtener las soluciones de la Figura 7 se han escogido los valores iniciales de los centroides de manera aleatoria.

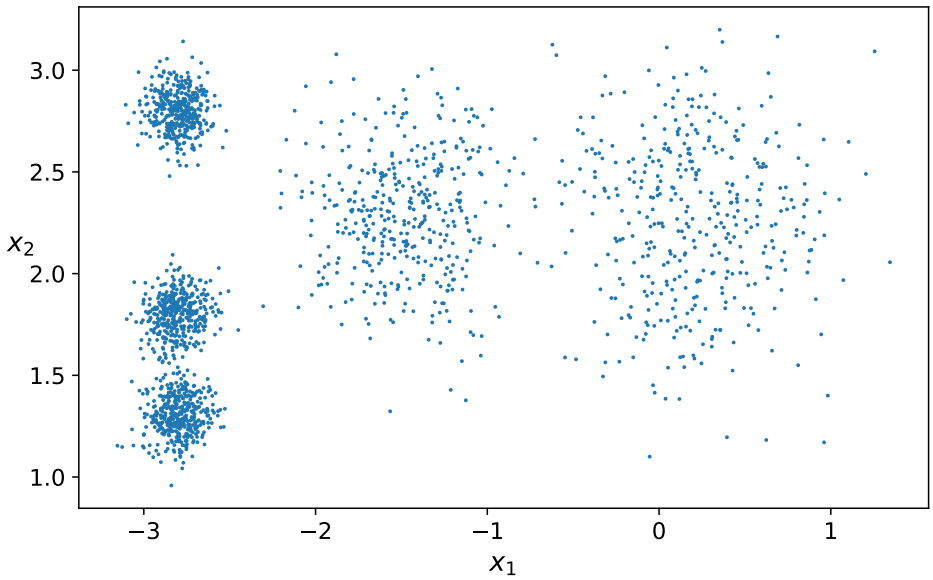


Figura 5: Conjunto de datos formado por la unión de 5 grupos

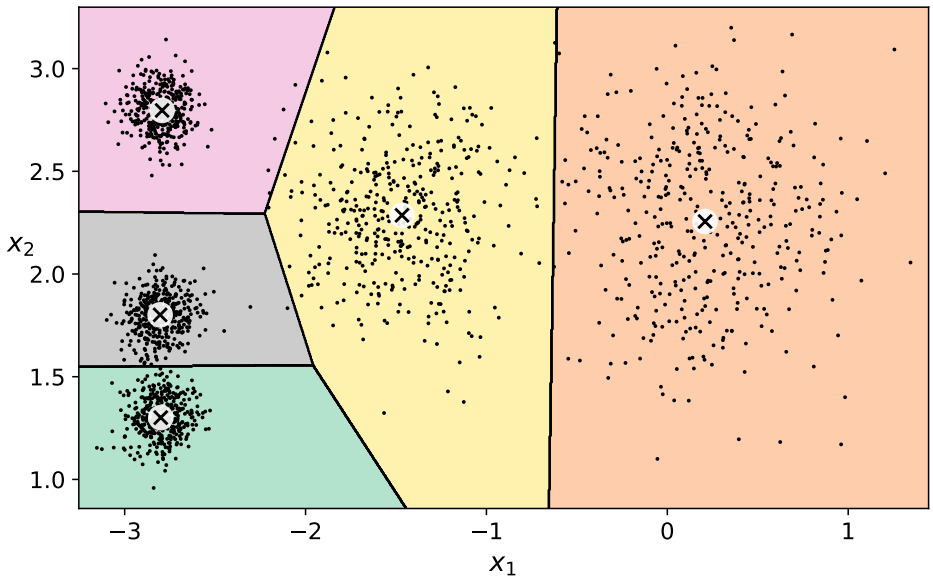


Figura 6: Solución óptima global del clustering con K-medias escogiendo los centroides de manera aleatoria

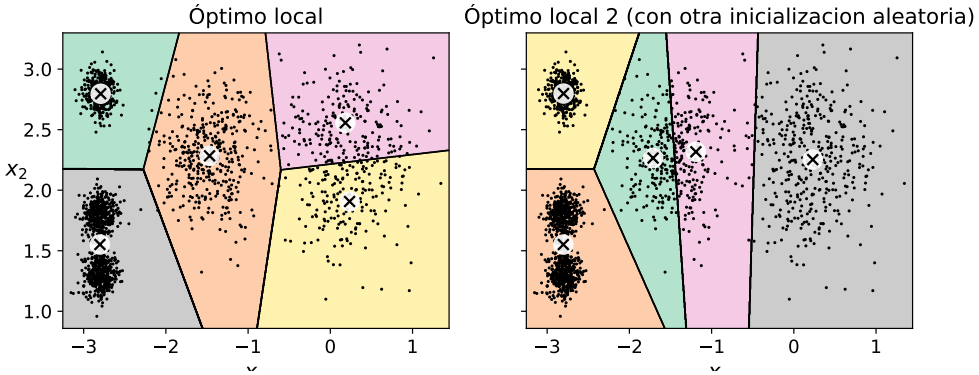


Figura 7: Soluciones con óptimos locales de clustering con K-medias escogiendo los centroides de manera aleatoria

Métodos de inicialización de los centroides

Para evitar obtener óptimos locales es adecuado aplicar algún metodo o estrategia a la hora de escoger los valores iniciales de los centroides en lugar de escogerlos aleatoriamente. Si se tiene alguna idea de más o menos cuales son los centroides se pueden introducir manualmente. No obstante, en muchas ocasiones esto no es posible y es necesario escogerlos empleando alguna estrategia.

Una de las estrategias posibles es escoger varios centroides iniciales de manera aleatoria, realizar el algoritmo iterativo para construir los *clusters* y escoger aquel que proporcione un mejor resultado. Para valorar qué *clustering* es mejor existe una métrica denominada inercia que mide las distancias medias al cuadrado entre cada observación y su centroide. Cuanto menor sea este valor, mejor será el *clustering* realizado. Esta es la técnica que emplea por defecto la clase *KMeans* de la librería *scikit-Learn*. En esta librería, además, los valores de los centroides iniciales se escogen de entre las distintas observaciones, pero de tal modo que las observaciones que se escogen como centroides iniciales estén suficientemente alejadas entre sí.

Encontrar el número óptimo de clusters K

Una vez que ya sabemos cómo funciona el método de K-medias queda responder otra pregunta a la hora de aplicar el método. ¿Cuántos grupos creo? Para responder a esta pregunta no podemos basarnos únicamente en el valor de la métrica de la inercia, ya que esta suele disminuir a medida que se aumenta el número de *clusters*. Sin embargo, sí que podemos ver cómo varía ésta con el número de *clusters* y seleccionar el valor al partir del cual se forma un codo (de forma similar a como se escogían el número de componentes principales). En la Figura 8 se muestra la aplicación del método del codo para el ejemplo anterior, donde se ha obtenido la inercia para K = 2 hasta 10 (en la Figura se muestran los valores de inercia para K = 2,…,10). Aunque sabemos que el número de grupos reales es 5, se observa que el cambio de tendencia se produce con K = 4, por lo que este sería el número de grupos óptimos si seguimos dicho método. Así pues, en la Figura 9 se muestra cual sería la solución óptima con K = 4.

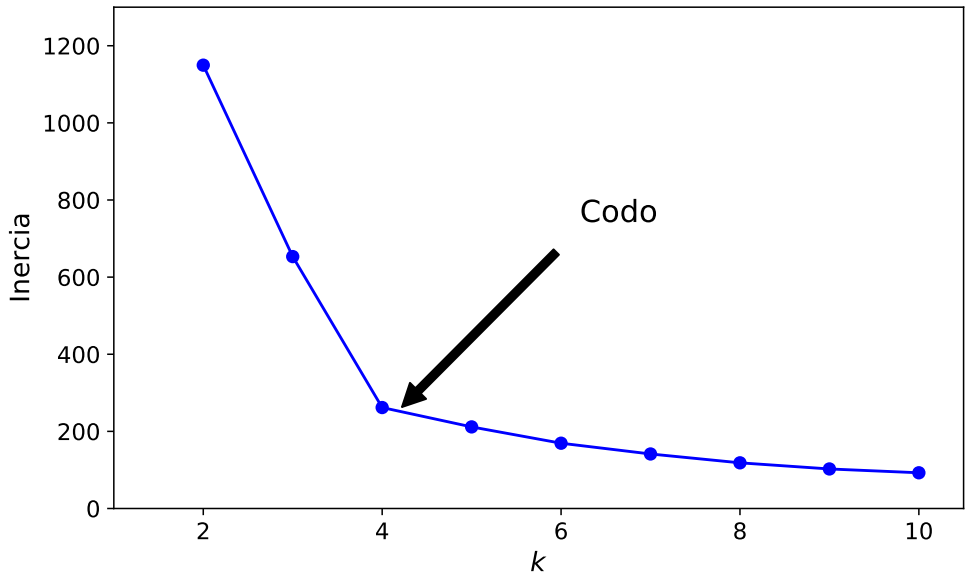


Figura 8: Elegir el número de clusters mediante el método gráfico del codo

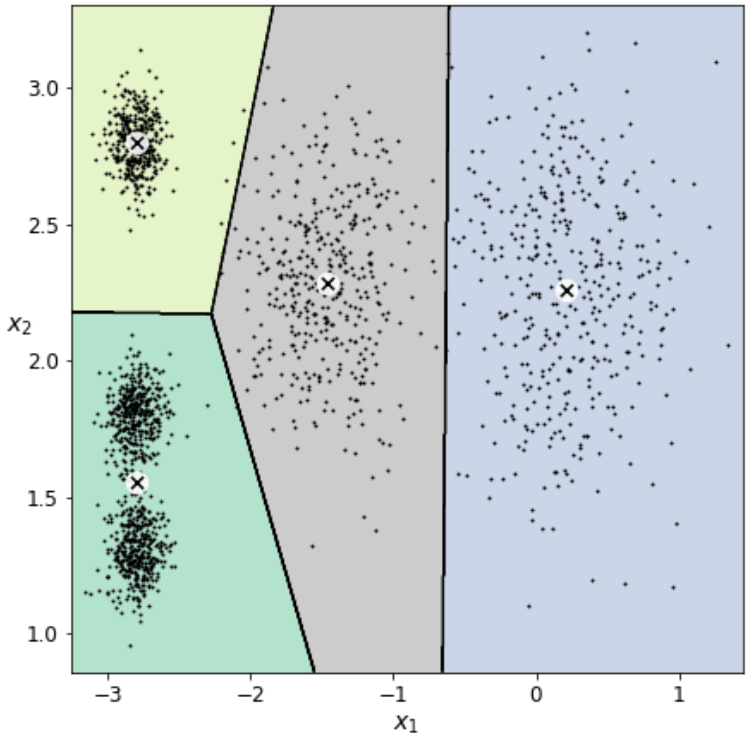


Figura 9: Solución óptima global del clustering con K-medias escogiendo K = 4

Material audiovisual

Accede al vídeo: Componentes Principales

9.7 Referencias bibliográficas

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2 edition.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning : with applications in R*. New York : Springer, [2013] ©2013.

Peña, D. (1987). *Regresión y diseño de experimentos*.

9.8 Ejercicios resueltos

Ejercicio 1.

Aplica el método de componentes principales en el conjunto de datos *mb.txt* y escoge las *d* primeras componentes principales.

a) Que garanticen un 95 % de la varianza original de los datos.

b) Por el método gráfico del codo.

c) Por el método de Kaiser.

Solución

El conjunto de datos *mb.csv* se puede descargar en la plataforma en el apartado de documentación. Está compuesto por 54 medidas de distintos metabolitos en sangre a 1175 individuos.

```
#cargar librerías ---------------------------
import pandas as pd
import matplotlib  as  mpl
import matplotlib.pyplot  as  plt
from sklearn.cluster  import  KMeans
from sklearn.cluster  import  MiniBatchKMeans
import numpy  as  np
# cargar base----------------------------------------------------
mb = pd.read_csv("datasets/mb.csv")
# info de la base-----------------------------------------------
print(mb.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1175 entries, 0 to 1174
## Data columns (total 54 columns):
##  #   Column                Non-Null Count  Dtype
## ---  ------                --------------  -----
##  0   alanine               1175 non-null   float64
##  1   creatin.phosphate     1175 non-null   float64
```

```
##  2   creatine              1175 non-null   float64
##  3   cysteine              1175 non-null   float64
##  4   glutamine             1175 non-null   float64
##  5   n.acetylglutamine     1175 non-null   float64
##  6   proline               1175 non-null   float64
##  7   tryptophan            1175 non-null   float64
##  8   tyrosine              1175 non-null   float64
##  9   isoleucine            1175 non-null   float64
##  10  leucine               1175 non-null   float64
##  11  valine                1175 non-null   float64
##  12  fa.ch2ch2co           1175 non-null   float64
##  13  fa.ch2ch3             1175 non-null   float64
##  14  fa.ch2n               1175 non-null   float64
##  15  fa.ch3                1175 non-null   float64
##  16  fa.chch2ch            1175 non-null   float64
##  17  isobutyrate           1175 non-null   float64
##  18  albumin               1175 non-null   float64
##  19  creatinine            1175 non-null   float64
##  20  citrate               1175 non-null   float64
##  21  lactate               1175 non-null   float64
##  22  pyruvate              1175 non-null   float64
##  23  acetate               1175 non-null   float64
##  24  acetone               1175 non-null   float64
##  25  hydroxybutyrate.3     1175 non-null   float64
##  26  ethanol               1175 non-null   float64
##  27  isopropanol           1175 non-null   float64
##  28  methanol              1175 non-null   float64
##  29  trimethylamines       1175 non-null   float64
##  30  phenylpropionate      1175 non-null   float64
##  31  O.phosphoethanolamine 1175 non-null   float64
##  32  total.c               1175 non-null   float64
##  33  vldl.c                1175 non-null   float64
```

```
##  34  ldl.c                1175 non-null   float64
##  35  idl.c                1175 non-null   float64
##  36  hdl.c                1175 non-null   float64
##  37  total.tg             1175 non-null   float64
##  38  vldl.tg              1175 non-null   float64
##  39  ldl.tg               1175 non-null   float64
##  40  idl.tg               1175 non-null   float64
##  41  hdl.tg               1175 non-null   float64
##  42  total.vldl           1175 non-null   float64
##  43  large.vldl           1175 non-null   float64
##  44  medium.vldl          1175 non-null   float64
##  45  small.vldl           1175 non-null   float64
##  46  total.ldl            1175 non-null   float64
##  47  large.ldl            1175 non-null   float64
##  48  medium.ldl           1175 non-null   float64
##  49  small.ldl            1175 non-null   float64
##  50  total.hdl            1175 non-null   float64
##  51  large.hdl            1175 non-null   float64
##  52  medium.hdl           1175 non-null   float64
##  53  small.hdl            1175 non-null   float64
## dtypes: float64(54)
## memory usage: 495.8 KB
## None
```

Se aplica el método de las componentes principales empleando la clase PCA de scikit-learn sin indicar ningún argumento. Por lo tanto se generarán tantas componentes principales como dimensiones originales se tenían (en este caso 54).

```
# se aplica el PCA con sklearn----------------------------------
from sklearn.decomposition import PCA
# crear clase PCA con n componentes-----------------------------
pca = PCA()
# obtener la proyeccion de la matriz mb en las comp principales--
```

```
mb_PC = pca.fit_transform(mb)
# comprobamos la dim de mb_PC-----------------------------------
mb_PC.shape
```

```
## (1175, 54)
```

a) Escoger el número de componentes principales que garanticen un 95 % de la varianza original de los datos.

Para ver qué porcentaje de la varianza original explica cada componente principal podemos hacer uso del método *.explained_variance_ratio_* contenido en la clase *PCA*.

```
print(pca.explained_variance_ratio_)
```

```
## [6.54113751e-01 1.18315153e-01 8.57161065e-02 4.49026558e-02
##  2.40614153e-02 1.57406515e-02 1.19984404e-02 1.07629315e-02
##  7.25341646e-03 5.09067583e-03 4.14285028e-03 2.86864343e-03
##  2.45528447e-03 1.87532558e-03 1.66420326e-03 1.51093331e-03
##  1.29703833e-03 1.15389535e-03 8.64402585e-04 8.08919830e-04
##  5.22737224e-04 4.35979353e-04 3.79831616e-04 3.39089074e-04
##  2.91621961e-04 2.25647457e-04 1.92138875e-04 1.40931503e-04
##  1.20856631e-04 1.16412240e-04 9.12743328e-05 7.54115534e-05
##  6.91451154e-05 5.60639983e-05 4.88777554e-05 3.89308594e-05
##  3.46320905e-05 3.20654047e-05 2.95220832e-05 2.72828096e-05
##  2.34086263e-05 1.87246371e-05 1.62239037e-05 1.52212230e-05
##  1.21886973e-05 1.12322939e-05 9.69063850e-06 7.50877781e-06
##  6.46432044e-06 4.32411037e-06 3.67087520e-06 2.56866886e-06
##  2.47900852e-06 1.14812962e-06]
```

Se observa cómo la primera componente principal es la que más varianza original explica, después la segunda y así sucesivamente hasta llegar a la componente quincuagésima cuarta, que es la que menos varianza original del conjunto de datos captura.

A continuación, se comprueba que efectivamente la suma de la varianza de todas las componentes principales es de 1 (el 100 %) y cuántas componentes principales serían necesarias para capturar el 0.95 (95 %) de la varianza original. Para obtener estos resultados se emplea la función *sum* y el método *cumsum*.

```
# suma de las varianzas de todas las componentes principales-----
print(round(sum(pca.explained_variance_ratio_), 5))
# acumulado de la varianza de todas las componentes principales--
```

```
## 1.0
```

```
print(pca.explained_variance_ratio_.cumsum())
```

```
## [0.65411375 0.7724289  0.85814501 0.90304767 0.92710908 0.94284973
##  0.95484817 0.96561111 0.97286452 0.9779552  0.98209805 0.98496669
##  0.98742198 0.9892973  0.9909615  0.99247244 0.99376948 0.99492337
##  0.99578777 0.99659669 0.99711943 0.99755541 0.99793524 0.99827433
##  0.99856595 0.9987916  0.99898374 0.99912467 0.99924553 0.99936194
##  0.99945321 0.99952863 0.99959777 0.99965384 0.99970271 0.99974164
##  0.99977628 0.99980834 0.99983786 0.99986515 0.99988855 0.99990728
##  0.9999235  0.99993872 0.99995091 0.99996215 0.99997184 0.99997934
##  0.99998581 0.99999013 0.9999938  0.99999637 0.99999885 1.        ]
```

Se observa que con 7 componentes principales se es capaz de capturar más del 95 % (0.9548) de la varianza original. Una vez que se ha determinado el número de componentes principales se puede introducir como argumento *n_components* a la hora de crear la clase *PCA*.

```
# crear clase PCA con 7 componentes-----------------------------
pca_7 = PCA(n_components = 7)
# obtener la proyeccion de la matriz mb en las com principales---
mb_PC_7 = pca_7.fit_transform(mb)
# comprobamos la dim de mb_PC-----------------------------------
mb_PC_7.shape
```

```
## (1175, 7)
```

Alternativamente, podemos ahorrar los pasos de obtener todas las componentes principales y aplicar el método *cumsum*, si indicamos directamente en el argumento n_components el tanto por uno de la varianza original que se requiere capturar por las componentes principales.

```
# crear clase PCA con 0.95 varianza explicada--------------------
pca_7_bis = PCA(n_components = 0.95)
# obtener la proyeccion de la matriz mb en las comp principales--
mb_PC_7_bis = pca_7_bis.fit_transform(mb)
# comprobamos la dim de mb_PC-------------------------------------
mb_PC_7_bis.shape
# se obtiene el mismo resultado que mb_PC_7-----------------------
```

```
## (1175, 7)
```

Por último, reconvertimos el resultado obtenido, que era un *array* de *numpy*, a un *dataframe* de *pandas* utilizando la función *pd.DataFrame*. Así pues, hemos conseguido pasar de un conjunto de datos con dimensión 54 (mb) a otro con dimensión 7 (mb_PC_7_df).

```
# representacion conjunto de datos en componentes principales----
mb_PC_7_df = pd.DataFrame(data = mb_PC_7,
columns = ["PC1","PC2","PC3","PC4","PC5","PC6","PC7"])
# primeras observaciones del nuevo conjunto de datos-------------
print(mb_PC_7_df.head())
```

```
##         PC1       PC2       PC3       PC4       PC5       PC6
## 0 -1.306603 -0.034829  0.229271 -0.451673  0.280763 -0.211506
## 1  0.081369 -1.001265 -0.069101 -0.164533  0.321772 -0.220160
## 2  4.451763 -2.034792 -1.154947  0.372276 -0.755619  0.136551
## 3  2.773560  5.209073 -0.228254 -0.382472 -0.208599 -0.044350
```

```
## 4  0.399323 -0.036426  0.743607  1.228068  0.175579  0.250505
##         PC7
## 0 -0.170962
## 1  0.046286
## 2  0.365416
## 3 -1.104817
## 4 -0.358567
```

b) Escoger el número de componentes principales por el método gráfico del codo.

En este caso, se representa gráficamente la proporción de varianza acumulada y se decide escoger aquella a partir de la cual no se aumenta considerablemente la varianza capturada. La varianza acumulada se presenta en la Figura 10.

```
# representacion conjunto de datos en componentes principales----
plt.plot(pca.explained_variance_ratio_.cumsum())
plt.show()
```

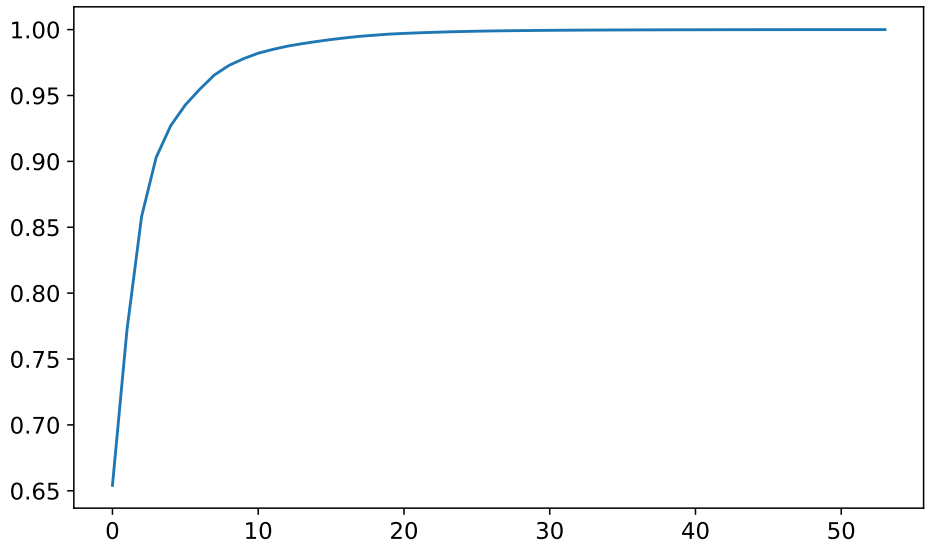


Figura 10: Varianza capturada acumulada por las componentes principales

Este método tiene el inconveniente de que su interpretación varía dependiendo del

observador pero puede ser útil para hacernos una primera idea de cuál es el número de componentes principales adecuado. En este caso se observa que a partir de 10 componentes principales no se consigue explicar muchas varianza adicional. Así pues, de modo similar al apartado anterior, se ajusta un objeto de clase PCA con el argumento *n_components = 10* y posteriormente se transforma en un dataframe.

```
pca_10 = PCA(n_components = 10)
mb_PC_10 = pca_10.fit_transform(mb)
# representacion conjunto de datos en componentes principales----
mb_PC_10_df = pd.DataFrame(data = mb_PC_10
              , columns = ["PC1","PC2","PC3","PC4","PC5","PC6",
              "PC7","PC8","PC9","PC10"])
# primeras observaciones del nuevo conjunto de datos------------
print(mb_PC_10_df.head())
```

```
##         PC1       PC2       PC3  ...       PC8       PC9       PC10
## 0 -1.306603 -0.034829  0.229271  ...  0.076528  0.342415  0.326570
## 1  0.081369 -1.001265 -0.069101  ... -0.023825 -0.301298  0.193147
## 2  4.451763 -2.034792 -1.154947  ...  0.767981  0.102946 -
0.160413
## 3  2.773560  5.209073 -0.228254  ...  1.025533 -0.169920 -
0.091074
## 4  0.399323 -0.036426  0.743607  ...  0.159043  0.257648 -
0.204155
##
## [5 rows x 10 columns]
```

c) Escoger el número de componentes principales según el criterio de Kaiser.

El criterio de Kaiser indica que hay que escoger aquellas componentes principales que expliquen una varianza superior a 1. Para ver qué valores cumplen podemos usar el método *.explained_variance_ratio_* contenido en la clase *PCA*.

```
# varianza explicada por las componentes principales-------------
print(pca.explained_variance_)
```

```
## [5.84872221e+00 1.05790845e+00 7.66425862e-01 4.01494632e-01
##  2.15143824e-01 1.40744171e-01 1.07283396e-01 9.62361612e-02
##  6.48560252e-02 4.55179986e-02 3.70430685e-02 2.56498178e-02
##  2.19537913e-02 1.67681207e-02 1.48803821e-02 1.35099273e-02
##  1.15973971e-02 1.03174920e-02 7.72900828e-03 7.23291227e-03
##  4.67402620e-03 3.89828546e-03 3.39624355e-03 3.03194635e-03
##  2.60752176e-03 2.01761435e-03 1.71799921e-03 1.26013130e-03
##  1.08063293e-03 1.04089365e-03 8.16124437e-04 6.74288266e-04
##  6.18257255e-04 5.01293164e-04 4.37037767e-04 3.48098143e-04
##  3.09660937e-04 2.86711057e-04 2.63970086e-04 2.43947744e-04
##  2.09306948e-04 1.67425315e-04 1.45065145e-04 1.36099730e-04
##  1.08984568e-04 1.00432940e-04 8.66483123e-05 6.71393247e-05
##  5.78003665e-05 3.86637955e-05 3.28229291e-05 2.29676116e-05
##  2.21659187e-05 1.02659380e-05]
```

En este caso se observa que sólo 2 componentes principales cumplen el criterio de Kaiser. Así pues, se escogerían únicamente las 2 primeras componentes principales. Del mismo modo que en los apartados anteriores se vuelve a ajustar la clase *PCA*, pero con el argumento *n_components = 2* y se obtiene el nuevo conjunto de datos representado en 2 dimensiones.

```
pca_2 = PCA(n_components = 2)
mb_PC_2 = pca_2.fit_transform(mb)
# representacion conjunto de datos en componentes principales----
mb_PC_2_df = pd.DataFrame(data = mb_PC_2
             , columns = ["PC1","PC2"])
# primeras observaciones del nuevo conjunto de datos-------------
print(mb_PC_2_df.head())
```

```
##          PC1       PC2
## 0 -1.306603 -0.034829
## 1  0.081369 -1.001265
## 2  4.451763 -2.034792
## 3  2.773560  5.209073
## 4  0.399323 -0.036426
```

Ejercicio 2.

Obtén K clusters del conjunto **Iris** mediante el método de K-medias.

a) ¿Cómo has escogido el inicio de los centroides?.

b) ¿Cómo has escogido el número de clusters óptimos?

Nota: se puede utilizar la clase real (*.target*) del conjunto de datos para comprobar que se ha realizado correctamente el *clustering*.

Solución

El primer paso consiste en cargar el conjunto de datos Iris. Este conjunto de datos ya se ha visto en temas anteriores. Aunque principalmente se ha trabajado con la longitud y la anchura del pétalo, el conjunto de datos está formado por 4 variables: las 2 anteriores más la longitud y la anchura del sépalo de la flor. Cuenta además con 150 observaciones, de 3 tipos de flores diferentes.

```
# cargar datos Iris----------------------------------------------
from sklearn.datasets import load_iris
data = load_iris()
# capturar matriz de las variables de iris-----------------------
```

```
X = data.data
# nombres variables X
data.feature_names
# dimension de X
```

```
## ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
##  'petal width (cm)']
```

```
X.shape
# grupos reales (para comprobar que el clustering)---------------
```

```
## (150, 4)
```

```
y = data.target
```

Para poder escoger cuál es el número de grupo óptimo se va a ver gráficamente cómo varía la inercia de la solución óptima para distintos K números de grupos (desde K=2 hasta K=10. Dentro de cada K se escogerá la mejor solución de entre distintas inicializaciones posibles de los centroides para evitar obtener máximos locales. Para definir que se van a emplear distintas inicializaciones posibles se utiliza el argumento *n_init* de la clase *Kmeans*. Para probar los distintos K números de grupo se hace uso de un bucle *for*. Para poder pintar las inercias, éstas se guardan en el objero *inercias*.

```
# definir ajuste Kmedias-----------------------------------------
kmedias_k = [KMeans(n_clusters = k, random_state = 3,
n_init = 10).fit(X)
                for k in range(2, 11)]
inercias = [model.inertia_ for model in kmedias_k]

# pintar inercias------------------------------------------------
plt.figure(figsize=(8, 4.75))
plt.plot(range(2, 11), inercias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inercia", fontsize=14)
```

```
plt.annotate('Codo',
             xy=(3, inercias[1]),
             xytext=(0.55, 0.55),
             textcoords='figure fraction',
             fontsize=16,
             arrowprops=dict(facecolor='black', shrink=0.1)
            )
plt.axis([1, 10.5, 0, 300])
```

```
## (1.0, 10.5, 0.0, 300.0)
```

```
plt.show()
```

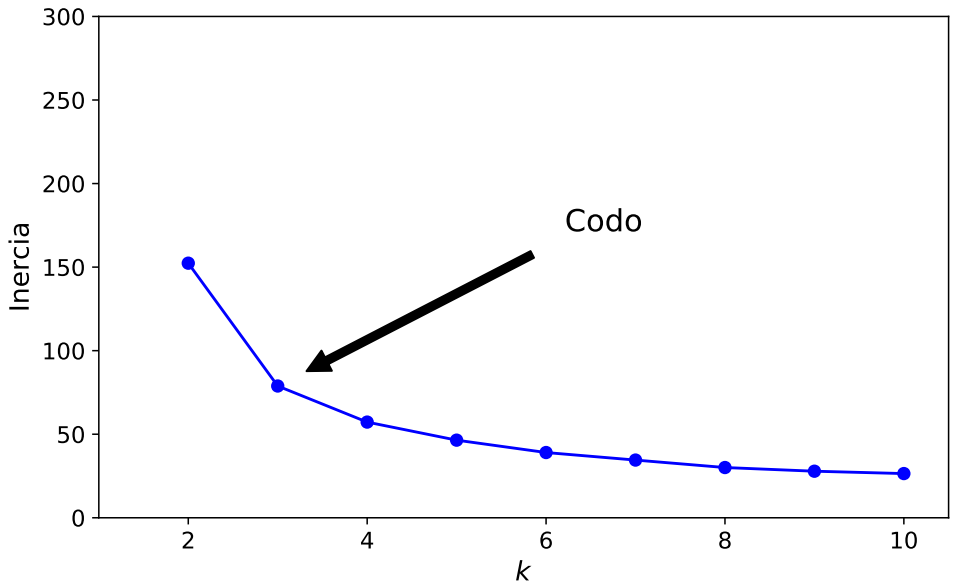


Figura 11: Método del codo para escoger K en el conjunto de datos Iris

En la Figura 11 se observa que el codo se forma a partir de 3 grupos, por lo que estos serán los grupos que se van a escoger. Además, se puede obtener el grupo en el que se ha dividido a cada observación mediante el método *.fit_predict* del objeto donde se ha realizado el ajuste de K medias y compararlo con las clases reales del conjunto de datos.

```
# definir ajuste Kmedias con K = 3------------------------------
kmedias_3 = KMeans(n_clusters = 3,
random_state = 3, n_init = 10).fit(X)
# calcular prediccion-------------------------------------------
y_pred = kmedias_3.fit_predict(X)
# clases del clustering----------------------------------------
print(y_pred)
# clases reales------------------------------------------------
```

```
## [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1
##  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1
##  2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1]
```

```
print(y)
```

```
## [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
##  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Se observa que todas las flores de la clase 0 se clasifican correctamente, aunque para algunas de las flores de las clases 1 y 2 existen discrepancias entre los valores predichos mediante el *clustering* y los valores reales. Esto se debe a que estas 2 clases tienen longitudes y anchuras de pétalo y sépalo similares.

Ejercicio 3.

En este ejercicio se va a emplear el conjunto de datos que ya se ha presentado en el ejercicio 1.

a) Obtén K-clusters del conjunto de datos *mb* mediante el método de las K-medias. Explica el método empleados para escoger el número de clusters óptimo.

b) Obtén K-clusters pero del conjunto de datos mb_PC que has obtenido en el ejercicio 1 después de aplicar el análisis por componentes principales.

Solución

a) De modo similar al ejercicio 2 se va a realizar un ajuste por K medias del conjunto de datos *mb*. En este caso, y al tener 1175 individuos, se va a comprobar el número de posibles grupos hasta K = 25.

```
# definir ajuste Kmedias para la base mb------------------------
kmedias_k_mb = [KMeans(n_clusters = k, random_state = 3,
n_init = 10).fit(mb)
                for k in range(2, 26)]
inercias_mb = [model.inertia_ for model in kmedias_k_mb]

# pintar inercias-----------------------------------------------
plt.figure(figsize=(8, 4.75))
plt.plot(range(2, 26), inercias_mb, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inercia", fontsize=14)
plt.annotate('Codo',
             xy=(4, inercias_mb[3]),
             xytext=(0.5, 0.5),
```

```
            textcoords='figure fraction',
            fontsize=16,
            arrowprops=dict(facecolor='black', shrink=0.2)
           )
plt.axis([1, 25.5, 0, 8000])
```

```
## (1.0, 25.5, 0.0, 8000.0)
```

```
plt.show()
```

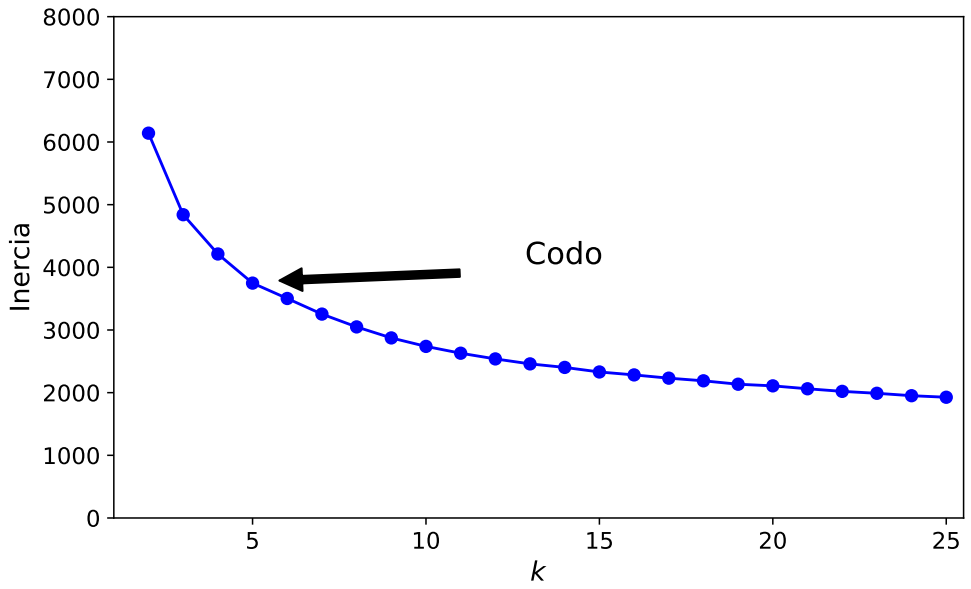


Figura 12: Método del codo para escoger K en el conjunto de datos mb

En la Figura 12 se observa que el codo se produce para K = 5. Así pues se vuelve a ajustar una clase *KMeans* fijando K = 5. Con este número de grupos se obtienen los resultados que se muestran a continuación. Se emplea la función *np.unique* sobre las predicciones para obtener la tabla de frecuencias de cada uno de los grupos. Se observa que el grupo con más observaciones es el grupo 4 que contiene a 419 observaciones mientras que el grupo con menos observaciones es el grupo 3 que únicamente contiene 25 observaciones.

```
# definir ajuste Kmedias con K = 5-----------------------------
kmedias_5_mb = KMeans(n_clusters = 5,
random_state = 3, n_init = 10).fit(mb)
y_pred = kmedias_5_mb.fit_predict(mb)
# clases del clustering----------------------------------------
unique, counts = np.unique(y_pred, return_counts = True)
dict(zip(unique, counts))
# clases reales------------------------------------------------
```

```
## {0: 368, 1: 218, 2: 145, 3: 25, 4: 419}
```

b) Ahora se repite el proceso anterior pero para el conjunto de datos *mb_PC_7_df*, obtenido de aplicar el análisis de componentes principales sobre el conjunto de datos *mb*.

```
# definir ajuste Kmedias para la base mb-----------------------
kmedias_k_mb_PC = [KMeans(n_clusters = k, random_state = 3,
n_init = 10).fit(mb_PC_7_df)
                for k in range(2, 26)]
inercias_mb_PC = [model.inertia_ for model in kmedias_k_mb_PC]

# pintar inercias----------------------------------------------
plt.figure(figsize=(8, 4.75))
plt.plot(range(2, 26), inercias_mb_PC, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inercia", fontsize=14)
plt.annotate('Codo',
             xy=(4, inercias_mb_PC[3]),
             xytext=(0.5, 0.5),
             textcoords='figure fraction',
             fontsize=16,
             arrowprops=dict(facecolor='black', shrink=0.2)
            )
```

```
plt.axis([1, 25.5, 0, 8000])
```

```
## (1.0, 25.5, 0.0, 8000.0)
```

```
plt.show()
```

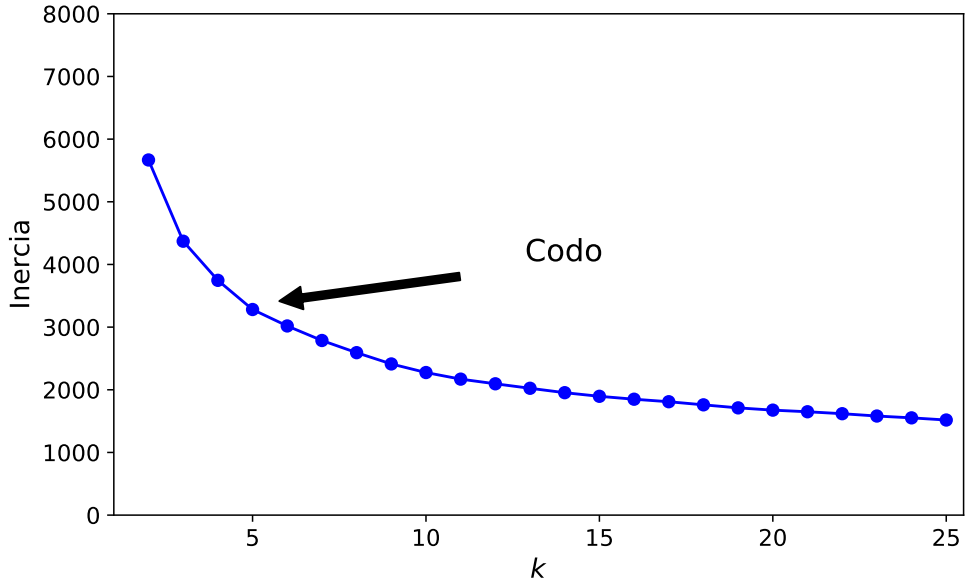


Figura 13: Método del codo para escoger K en el conjunto de datos mb_PC

En la Figura 13 se observa que el codo también se produce para K = 5. Así pues se vuelve a ajustar una clase *KMeans* fijando K = 5. Con este número de grupos se obtienen los resultados que se muestran a continuación. Se emplea la función *np.unique* sobre las predicciones para obtener la tabla de frecuencias de cada uno de los grupos. Se observa que el grupo con más observaciones es el grupo 2 que contiene a 440 observaciones mientras que el grupo con menos observaciones es el grupo 4 que únicamente contiene 26 observaciones.

```
# definir ajuste Kmedias con K = 5------------------------------
kmedias_5_mb_PC = KMeans(n_clusters = 5,
random_state = 3, n_init = 10).fit(mb_PC_7_df)
y_pred = kmedias_5_mb_PC.fit_predict(mb_PC_7_df)
# clases del clustering-----------------------------------------
```

```
unique, counts = np.unique(y_pred, return_counts=True)
dict(zip(unique, counts))
# clases reales-----------------------------------------------------
```

```
## {0: 384, 1: 179, 2: 440, 3: 146, 4: 26}
```

Se comprueba que el número de grupos adecuado es el mismo para el conjunto de datos original y para el conjunto de datos una vez aplicado el análisis de componentes principales, como era de esperar.