ALGORITMOS NO SUPERVISADOS

A diferencia de los algoritmos supervisado en los que contamos con una target, una variable *y,* que sería la evaluación de cada observación, en los algoritmos no supervisados no contamos con dicha variable, lo que significa que los datos no están *entrenados*. Si tenemos en cuenta que los algoritmos supervisados se basan en ajustar las variables predictoras a la variable target, es fácil deducir que en los problemas en los que no exista esa variable target no podremos utilizar los algoritmos supervisados. De esa necesidad nacen los algoritmos no supervisados.

Existen dos grandes tipos de algoritmos no supervisados:

- **Algoritmos de clusterización**: Algoritmos cuyo objetivo es agrupar las observaciones en grupos que tengan algún tipo de característica común. En clase vimos el k-means y el DBSCAN de los que luego hablaremos.
- **Algoritmos de reducción de la dimensionalidad**: En este caso el objetivo es alterar el eje de coordenadas de nuestros datos para contener la mayor cantidad de información posible en la menor cantidad de variables posible. En clase vimos el PCA.

Algoritmos de clusterización

Todos los algoritmos de clusterización tienen algo en común, que su objetivo es agrupar los puntos que más se parecen entre sí. Será en la definición de *entre sí* y en la definición de *parecer* en lo que se diferencien todos los algoritmos de clusterización.

K-means

1. Inicializo k centroides de manera aleatoria
2. Calculo cada punto de que centroide esta más cerca
3. Recalculo la situación del centroide siendo ahora la posición media de todos los puntos asignados en el paso 2.
4. Repito los pasos 2-3 hasta que se estabilice (ningún punto cambie de centroide)

Encontrar el K optimo

Hemos visto dos formas de encontrar el k óptimo de un conjunto de datos

- **Intertia + elbow**: *Inertia* nos suma la distancia de cada punto a su centroide. Como esta suma siempre va a ser menor cada vez que se amplíe la *k* por sí sola no podemos sacar información de que *k* es mejor. Pero podemos guiarnos por la técnica de elbow (codo en ingles), que se basa en escoger la *k* donde la gráfica de

inertias para cada *k* dibuja un codo, es decir, donde la mejora de *inertia* de un k al siguiente se reduce mucho.
- **Silhouette score**: Esta técnica al contrario de la anterior nos da una métrica numérica para poder evaluar el número de *k*.

Evaluación de k-means

El k-means tiene un problema importante, es muy sensible a la inicialización de los centroides y diferentes inicializaciones no tienen porque converger en un mismo resultado, por lo que conviene poder evaluar los modelos para saber que inicialización es mejor. En este caso la *inertia* sí que nos puede ser de utilidad, ya que estamos comparando diferentes modelos pero todos ellos con la misma cantidad de centroides.

Problemas del K-means

- Es sensible a la inicialización aleatoria de los centroides
- Sensible a outliers
- Hay que definir de antemano la cantidad de clusters que queremos
- Al basarse en distancias euclideas podemos tener problemas si no tenemos los datos normalizados o cuando crezca mucho las dimensiones de nuestro problema

Visualización del k-means: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

DBSCAN

1. Definimos epsilon (distancia desde un punto para considerarlo vecino) y minPoints (número mínimo de vecinos para considerar a un punto nucleo)
2. Buscamos todos los vecinos para todos los puntos y los puntos que tengas más vecinos que minPoints lo marcamos como núcleo.
3. Cada punto núcleo y todos sus vecinos lo asignamos a un cluster, si no esta asignado previamente a ninguno. Recursivamente vemos añadiendo a sus vecinos núcleo.

El DBSCAN da solución a los problemas que nos encontramos en el k-means, aunque también tiene sus propios problemas.

El principal problema del DBSCAN es que es bastante sensible a la inicialización del epsilon y los minPoints.

Visualización del DBSCAN: https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Si tenéis curiosidad en otro tipo de algoritmos de clusterización en este artículo de medium podéis ver algunos:

https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Algoritmos de reducción de la dimensionalidad

Una de las técnicas de preprocesado de los datos para modelos de machine learning es la reducción de la dimensionalidad, que no es más que la reducción de variables en una colección de datos.
Algunos de los motivos por los que nos interesaría reducir las dimensiones van desde la identificación y eliminación de variables irrelevantes, hasta la mejora del rendimiento computacional, lo que se traduce en un ahorro en coste y tiempo.

PCA

La idea básicamente es construir una dimensión (o variable) nueva a partir de la fusión de dos ya existentes. El método PCA (Análisis de Componentes Principales) permite por lo tanto "condensar" la información aportada por múltiples variables en solo unas pocas componentes. Para ello realizamos lo siguiente:

1. Centramos nuestros datos.
2. Obtenemos la matriz de covarianza de nuestros datos.
3. Realizamos transformaciones lineales a nuestra matriz, de la cual obtenemos los vectores propios (eigenvectors) y sus respectivos valores propios (eigenvalues).
4. Aquellos valores más altos, nos determinan una mayor varianza, y por tanto, nos indican dónde mantenemos más información.
5. A continuación, proyectamos nuestros puntos sobre el vector seleccionado, el cual será nuestra componente principal.
6. Esta nueva dimensión, será con la que nos quedemos, y es el resultado de las dos anteriores.

A tener en cuenta:

- Dado que PCA trabaja con varianzas, debemos tener nuestros datos en una misma escala.
- Por el mismo motivo, PCA es muy sensible a outliers.
- Debemos tener en cuenta cuánta información presente en el set de datos original se pierde al realizar las proyecciones. Para medirlo, utilizaremos la varianza explicada de cada componente principal.
- Lo interesante es utilizar el número mínimo de componentes que resultan suficientes para explicar nuestros datos.