Encuentro 22: Modelos de Clustering

1- Resumen

1.1 - Modelos de Clasificación

1. **Árboles de decisión:** es similar a un diagrama de flujo y tiene una estructura formada por nodos, hojas y ramas. Las instancias atraviesan esta estructura para ser clasificadas. Cada nodo realiza una pregunta a las instancias y según la respuesta, va a una u otra rama. Las hojas se encuentran en las extremidades del diagrama y cada una representa una etiqueta. Optimizar las preguntas que se realizan en los nodos es optimizar el modelo.
2. **k-vecinos más cercanos:** dada una nueva instancia, el modelo busca las *k* instancias más cercanas del dataset de entrenamiento. Luego, asigna la etiqueta según las etiquetas de estas *k* instancias más cercanas. La cantidad de datos en el radio a tener en cuenta es un hiperparámetro. El voto puede ser mayoritario o ponderado.
3. **Regresión Logística:** se ajusta una función sigmoide a los datos. Esa función se interpreta como la probabilidad de que las instancias sean positivas o negativas. El modelo se usa principalmente para clasificación binaria, aunque se puede expandir a problemas multiclase. Posteriormente, se define un umbral que determinará la frontera de decisión en la clasificación de las instancias. El umbral es un hiperparámetro y definirá la sensibilidad y exactitud del modelo. El nombre suele confundir.
4. **Support Vector Machines:** buscamos la frontera que mejor separa las clases. Se suele usar para problemas de clasificación binarios, pero existen estrategias para abordar problemas multiclase. Aplicable a problemas no linealmente separables gracias a los kernels.

Para profundizar en Modelos de Clasificación

Hasta ahora, entrenamos y evaluamos cada modelo individualmente. Pero tal vez podemos combinarlos, de forma tal que en este proceso obtengamos un modelo aún mejor que el mejor modelo individual. Un ensamble es, en principio, un conjunto de cosas, en general todas similares pero con pequeñas diferencias entre sí.

Segun Wikipedia:

“En estadística y Machine Learning, el método de ensamble usa múltiples algoritmos para obtener mejores predicciones que el que podría obtener con cualquiera de los algoritmos individualmente.”

En general, los modelos de ensamble se basan en la votación. Si tenemos muchos modelos juntos, la clasificación resultante es la que reciba más votos. Es decir, si nuestro ensamble consiste de 100 clasificadores, de los cuales 90 predicen la Clase A, y 10 la Clase B, la clasificación resultante es la Clase A. Aún mejor, si los modelos predicen **scores,** se puede hacer una votación ponderada, dándole más peso a aquellos clasificadores que están muy seguros de la etiqueta predicha.

Los modelos de ensamble se pueden hacer sobre cualquier modelo base, es decir, cualquiera de los que mencionamos más arriba. Pero lo más común es hacerlo sobre árboles de decisión, ya que es un modelo sencillo, relativamente rápido de entrenar y eficiente.

La pregunta, entonces, es: ¿cómo construimos estos ensambles? Existen varias técnicas para combinarlos, pueden ser a través de bagging, boosting y stacking.

1.2 - Under/Overfitting

Generalización del conocimiento

Como si se tratase de un ser humano, las máquinas de aprendizaje deberán ser capaces de generalizar conceptos. Supongamos que **vemos un perro Labrador por primera vez en la vida** y nos dicen *“eso es un perro”*. Luego nos enseñan un Caniche y nos preguntan: ¿eso es un perro? Diremos “No”, pues no se parece en nada a lo que aprendimos anteriormente. Ahora imaginemos que nuestro tutor nos muestra un libro con fotos de 10 razas de perros distintas. Cuando veamos una raza de perro que desconocíamos seguramente seremos capaces de reconocer al cuadrúpedo canino *al tiempo de poder discernir* en que un gato no es un perro, aunque sea peludo y tenga 4 patas.

Cuando entrenamos nuestros modelos computacionales con un conjunto de datos de entrada estamos haciendo que **el algoritmo sea capaz de generalizar un concepto** para que al consultarle por un nuevo conjunto de datos **desconocido** *éste sea capaz de sintetizarlo, comprenderlo y devolvernos un resultado fiable* dada su capacidad de generalización.

Si nuestros datos de entrenamiento son muy pocos nuestra máquina no será capaz de generalizar el conocimiento y estará incurriendo en *underfitting*. Este es el caso en el que le enseñamos sólo una raza de perros y pretendemos que pueda reconocer a otras 10 razas de perros distintas. El algoritmo no será capaz de darnos un resultado bueno por falta de “materia prima” para hacer sólido su conocimiento. También es ejemplo de “subajuste” cuando la máquina reconoce todo lo que “ve” como un perro, tanto una foto de un gato o un coche.

Por el contrario, si entrenamos a nuestra máquina con 10 razas de perros **sólo de color marrón** de manera rigurosa y luego enseñamos una foto de **un perro blanco**, nuestro modelo no podrá reconocerlo cómo perro por no cumplir exactamente con las características que aprendió (el color forzosamente debía ser marrón). Aquí **se trata de un problema de overfitting**.

Tanto el problema del ajuste “por debajo” como “por encima” de los datos son malos porque no permiten que nuestra máquina generalice el conocimiento y no nos darán buenas predicciones (o clasificación, o agrupación, etc.)

1- Cómo Detectabamos el over/Undefitting?

La realidad en machine learning (en general) es que no podemos saber que tan bien está funcionando hasta que no lo testeemos con nuevos datos.

Para solucionar esto, se dividía el set de datos total en un set de training (o entrenamiento) y un set de prueba (o testing)

Este método de dividir el conjunto de datos en train y test puede decirnos aproximadamente que tan bien reaccionaria nuestro modelo a datos que nunca ha visto.

En este punto, si nuestro modelo lo hace mucho mejor en el training set que en el test set, hay altas probabilidades de estar en overfitting

2- Como prevenirlo?

Obvio que detectar el overfitting nos ayuda, pero no soluciona el problema. Les dejo un par de opciones para prevenirlo que pueden investigar por su cuenta:

1- Cross Validation

2- Entrenar con mas datos

3- Early Stopping point

4- Regularización (L1 Ridge o L2 Lasso)

5- Modelos de ensamble

Adicionalmente les recomiendo que investiguen los siguientes temas:

- Señal, ruido y que relacion guardan con el overfitting.
- Bias and Variance. Tradeoff. (Sesgo y varianza en español)

1.3- Métricas

Profundicen con esta serie de artículos. https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

Hay miles y miles de maneras de evaluar un problema. Tenemos las métricas más clasicas, como las que estuvimos viendo la clase pasada, hasta métricas mas “custom”. En ese caso, esta métrica hecha a medida suele estar super alineada a algun KPI o métrica interna del negocio.

Entonces, vimos primero la matriz de confusión (que no es una métrica como tal, pero es fundamental)

		Actual Class	
		Cat	Non-Cat
Predicted Class	Cat	90	60
	Non-Cat	10	940

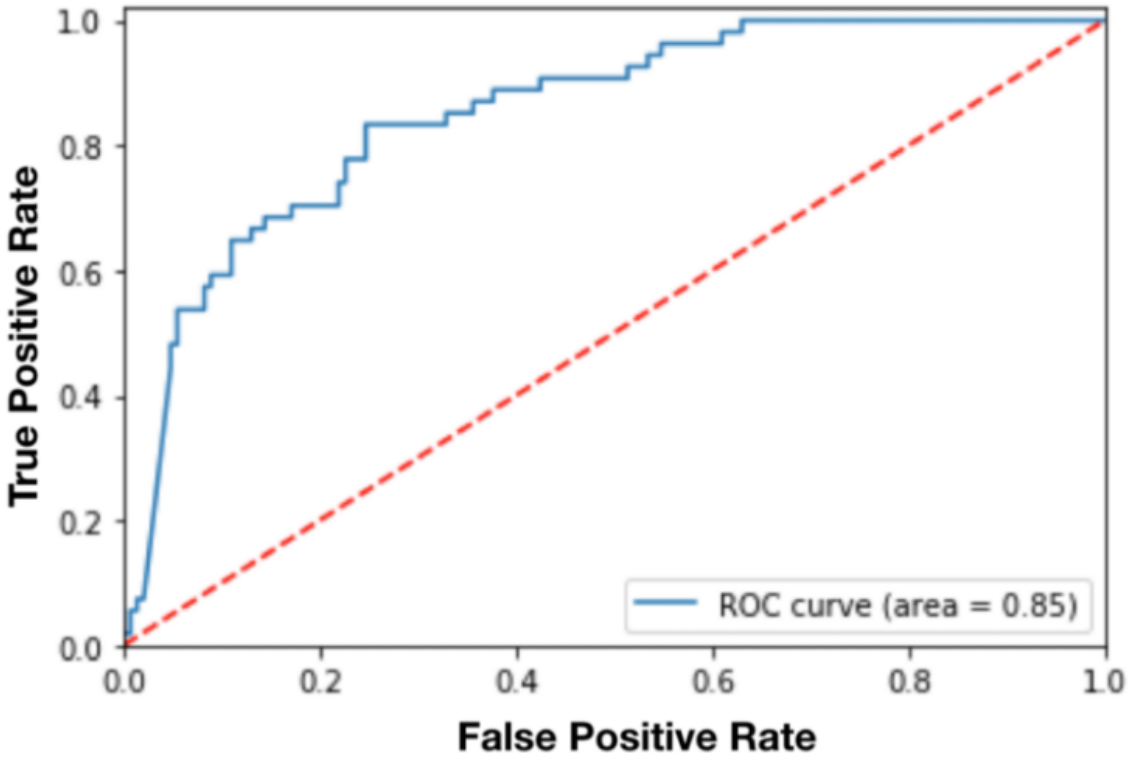
Después vimos el **Accuracy o exactitud.** Dijimos que era la métrica más simple que uno se puede imaginar ya que esta definido como el número de predicciones correctas, sobre el numero total de predicciones.

Les conté que hay muchos casos donde el accuracy o exactitud no viene a ser el mejor indicador del performance del modelo... Un caso clásico puede ser una distribución de clases imbalanceada, aka, una clase es mas frecuente que otra (¿Se les ocurre alguna? - Pista: Detección del Fraude). La precisión que es una métrica específica de cada clase nos puede dar mas granularidad sobre el funcionamiento del modelo a la hora de clasificar en clase X o Y.

Recall por ultimo se refiere a la cantidad que el modelo de machine learning es capaz de identificar. Es decir, sobre el total **REAL** de una etiqueta, cuantos somos capaces de predecir correctamente?

El valor F1 se utiliza para combinar las medidas de precision y recall en un sólo valor. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones.

Para profundizar aca, ver de que se tratan las métricas de Sensitivity y Specificity. Por otro lado una super importnate en clasificación binaria es el ROC curve.



1.4 - Optimización de HP

Este concepto se refiere a encontrar la combinación de hyperparámetros de un modelo tal que maximice o minimice la métrica de evaluacion.

Es decir, la combinacion de los hiperparametros de la clase DeccissionTreeeClassifier tal que maximice el accuracy.

O dicho de otro modo en un problema de regresión. Encontrar la combinación de hiperparámetros tal que minimice el error cuadrático medio.

Vimos como aplicarlo “manualmente” en python, donde vemos como cambia el performance variando solo UN hyperparámetro.

Acá si les dejo como tarea de profundizar que son los Grid Search? Que son los Randomized Search y busquen ventajas y desventajas de optimizar con uno y con el otro.

1.5 - Desafío.

Acá les dejo 8 casos de uso de Machine Learning. Todos corresponden al paradigma de aprendizaje supervisado, por lo que quiero que me digan para cada caso:

1. Que va a predecir el modelo?. Textualmente que es lo que quiere predecir. Numericamente que tipo de variable estamos midiendo? Cuantos resultados posibles existen?
2. Que tipo de problema es? Clasificación o Regresión?
3. En el caso de que sea clasificación. Es binaria o multiclase?
4. En el caso de que sea regresión. Es discreta o continua?

2 - Aprendizaje no supervisado. Modelos de Clustering

2.1 - Introducción al Aprendizaje no supervisado.

Link para libro : https://www.cienciadedatos.net/documentos/py20-clustering-con-python.html

No supervisado significa que no contamos o, más comúnmente, no utilizamos las etiquetas asociadas a nuestros datos. Es común decir que sirve para encontrar patrones en los datos, sin intervención (supervisión) humana.

La gran diferencia es que no hay un conocimiento a priori (en el sentido de las etiquetas asociadas). Con esto quiero decir quel resultado final no se acerca o asemeja a lo esperamos, por el hecho de que no sabemos cual es el resultado final o el resultado esperado.

Los algoritmos de Aprendizaje no Supervisados infieren patrones de un conjunto de datos sin referencia a resultados conocidos o etiquetados. Se suelen utilizar para descubrir la estructura subyacente de los datos.

Dentro del paradigma del aprendizaje no supervisado tenemos dos grupos, Clustering y Reducción de dimensionalidad.

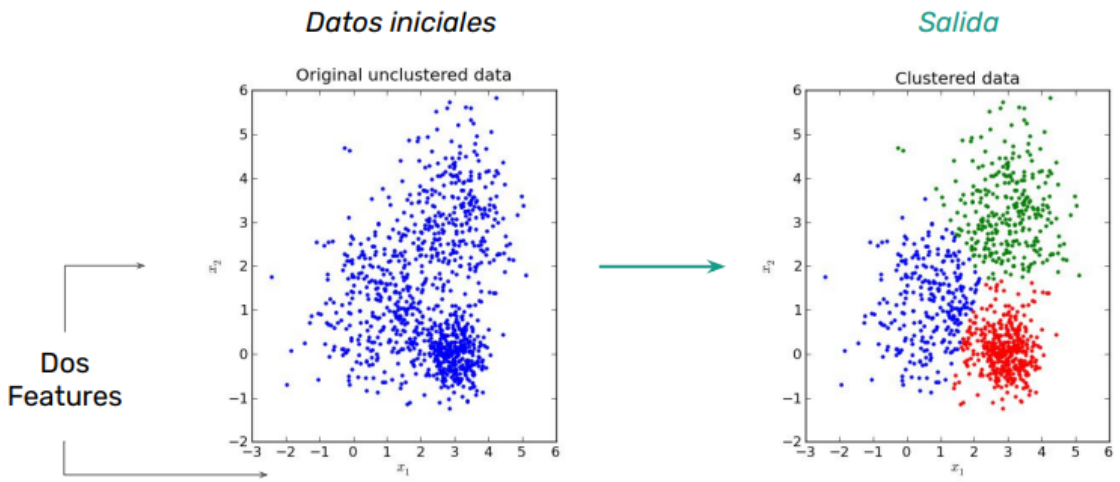
En esta clase nos vamos a adentrar en Clustering.

2.2 - Clustering

En Clustering, los datos se agrupan según características en común. ¿Qué quiere decir esto? ¡Veamos un ejemplo!

Imagina que contamos con un dataset de dos atributos y graficamos las instancias con gráfico de dispersión, obteniendo el gráfico de la izquierda que ves a continuación. Si tuvieras que elegir dos grupos dentro de estos datos, ¿cuáles elegirías? ¡Es una tarea bastante intuitiva!

Clustering consiste, precisamente, en elegir estos grupos (*clusters*). Una vez elegidos los grupos, puedes pintar los elementos que componen cada cluster de un color cada uno para diferenciarlos, obteniendo la figura de la derecha.



¡No importa si tenemos o no etiquetas! Si no las tenemos, clustering nos ayudará a agrupar las instancias según su similaridad y nos puede ayudar en el proceso de generar etiquetas para esos datos. Si las tenemos, clustering nos puede ayudar a corroborar que esas etiquetas *tienen sentido*. Algunos usos de clustering son:

• **Para desarrollar hipótesis.** Clustering es una de las primeras cosas que hacen muchos Data Scientist al encarar un proceso de exploración de un dataset. Si los clusters están compactos y bien separados, debería haber una razón que lo explique y es el trabajo del data scientist descubrir cuál es. Una vez que etiquetes cada grupo, puedes estudiar cuáles son sus diferencias y similitudes.

• **Para separar los datos.** Muchas veces contamos con datasets muy extensos, hasta con millones de instancias. En ocasiones puede ser una buena idea separar en clusters los datos, seleccionar el cluster que mejor representa las instancias que querés predecir y una vez hecho eso; entrenar tu modelo sobre ese cluster.

• **Para reducir el dataset.** Con el objetivo de reducir nuestro dataset, una técnica frecuente es la de utilizar los *centroides* como representación de cada cluster. De esta manera podemos contar con un dataset robusto y confiable, pero mucho más chico y procesable para las computadoras. El centroide representa el centro de cada cluster, te va a quedar mucho más claro después de leer K-mean.

• **Para detectar outliers:** Clustering es un buen primer paso para encontrar *outliers*. Estos pueden aparecer de dos maneras: dispersos y sin cercanía a ningún cluster o pueden estar agrupados entre ellos. Para detectar el segundo caso, es una buena práctica prestarle atención a pequeños clusters, alejados de los demás. Si se da ese caso, es probable que se trate de outliers agrupados.

Y algunas aplicaciones más prácticas en las que se usa Clustering son:

- Investigación de Mercado
- Sistemas de Recomendación
- Medicina
- Biología genética y especis
- etc, etc, etc

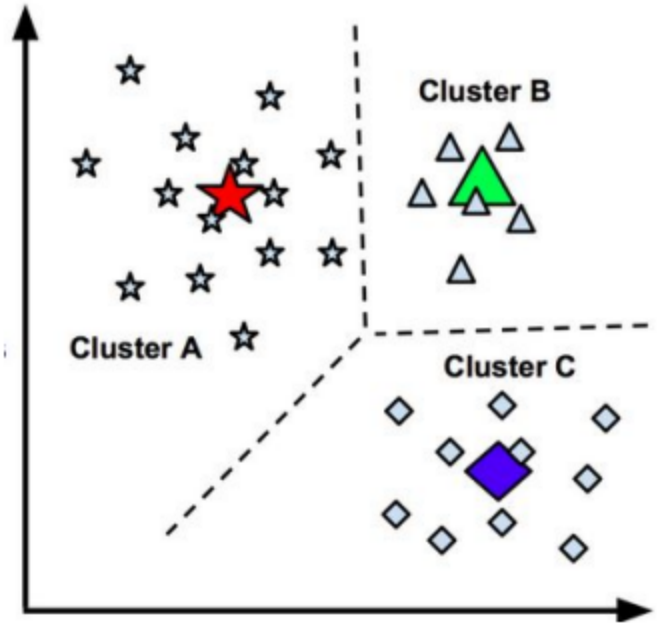
Dada la utilidad del *clustering* en disciplinas muy distintas (genómica, marketing...), se han desarrollado multitud de variantes y adaptaciones de sus métodos y algoritmos. Si bien hay muchas formas de clasificar los tipos de algoritmos de Clustering, me parece muy acertada la siguiente:

- *Partitioning Clustering*: este tipo de algoritmos requieren que el usuario especifique de antemano el número de *clusters* que se van a crear (*K-means*, *K-medoids*, *CLARA*).
- *Hierarchical Clustering*: este tipo de algoritmos no requieren que el usuario especifique de antemano el número de *clusters*. (*agglomerative clustering*, *divisive clustering*).
- Métodos que combinan o modifican los anteriores (*hierarchical K-means*, *fuzzy clustering*, *model based clustering* y *density based clustering*).

Para empezar con los modelos, vamos a ver el más simple de todos, se llama K-Means, no confundir con KNN

2.3 K-Means

En pocas palabras el algoritmo *K-means* (MacQueen, 1967) agrupa las observaciones en un número predefinido de *K clusters* de forma que, la suma de las varianzas internas de los clusters, sea lo menor posible.

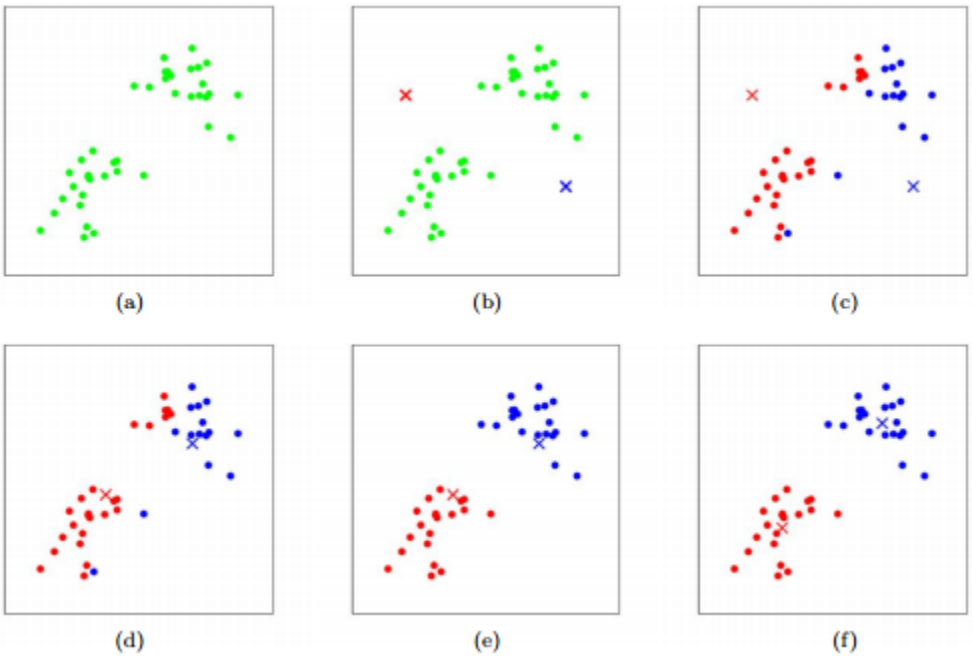


Ten en cuenta dos conceptos claves:

- El centro de cada cluster **(*centroide*)** es el promedio de todos los puntos pertenecientes a ese cluster.
- Cada punto está más cerca del centro de su cluster que de cualquier otro centroide.

Este algoritmo trabaja de manera iterativa, vuelve a repetir el proceso hasta encontrar la mejor solución al problema.

¿Como lo hace?



1. Partimos de los datos sin etiquetas.
2. Se inicializan los *k* centroides. Verás que en la secuencia que graficamos a continuación, elegimos 2 (esta elección no es siempre obvia). La ubicación inicial de los centroides puede ser aleatoria o con algún criterio. La cantidad de centroides va a determinar cuántos clusters vamos a obtener. Es decir, cada centroide está asociado a un cluster.
3. Se asigna cada instancia al centroide/cluster más cercano (el significado de “cercano” puede cambiar, es un hiperparámetro).
4. Actualizamos los centroides. La nueva posición de cada centroide es el promedio de las posiciones de las instancias en ese cluster. De acá viene el nombre *means*.
5. Repetimos pasos 3 y 4 hasta que la posición de los centroides ya no varíe.

El paso 4 es el que suele llamar más la atención. ¿Por qué debería funcionar? Como te contare un poco más adelante, esta técnica es un caso particular de un algoritmo conocido como Esperanza-Maximización (EM), que nos garantiza que bajo ciertas condiciones el proceso converge.

¿Qué cantidad de clusters elegir?

La pregunta de cuántos clusters tiene un dataset es fundamental. Sobre todo teniendo en cuenta que la falta de conocimiento sobre la estructura de nuestros datos es inherente a esta técnica.

Muchas veces podemos tener una noción de las poblaciones que queremos identificar -republicanos y demócratas; fanáticos de sus respectivos clubes o hábitos de consumos- y esto nos puede ayudar a determinar el número de clusters que vamos a formar. Pero muchas veces no tendremos una idea de si lo que buscamos son 2, 10, 100 o 1.000 clusters. ¿Se te ocurre alguna manera de encontrar el número óptimo?

Vamos a ver en el último tema de hoy, métricas de evaluación para poder elegir “criteriosamente” que es “mejor”

Esperanza- Maximización

Si K-means tuviera que evaluar todas las posibles combinaciones de clusters, la tarea sería exponencial a medida que la cantidad de instancias se incrementa. De esta manera, la eficiencia de procesamiento sería mucho menor que la actual -que no está de más agregar, es relativamente alta-. La eficiencia de k-means se da gracias a que utiliza un algoritmo llamado Expectation-maximization (EM). Este es muy utilizado en diversos contextos de Data Science.

Hay un problema que puede surgir al trabajar con el algoritmo EM: si bien el proceso de EM te asegura que cada paso clasifica mejor o igual que la clasificación anterior, esto no garantiza que la solución final será la mejor solución global. En determinados random seeds -llamamos random seed a los primeros centroides que se seleccionan aleatoriamente- el modelo ajusta mal.

Un ejemplo de un modelo de 4 clusters evidentemente mal agrupados:

```
In [6]: centers, labels = find_clusters(X, 4, rseed=0)
        plt.scatter(X[:, 0], X[:, 1], c=labels,
                    s=50, cmap='viridis');
```

La manera de subsanar esto (Scikit-Learn lo hace por ti) es utilizar varios Random Seeds y evaluar los resultados de cada solución. ¡Es similar a lo que pasaba con los mínimos locales en descenso por gradiente!

Ver como funciona el algoritmo de manera visual: https://www.youtube.com/watch?v=BVFG7fd1H30

Ventajas y Desventajas

K-means es uno de los métodos de *clustering* más utilizados. Destaca por la sencillez y velocidad de su algoritmo, sin embargo, presenta una serie de limitaciones que se deben tener en cuenta.

- Requiere que se indique de antemano el número de *clusters* que se van a crear. Esto puede ser complicado si no se dispone de información adicional sobre los datos con los que se trabaja. Se han desarrollado varias estrategias para ayudar a

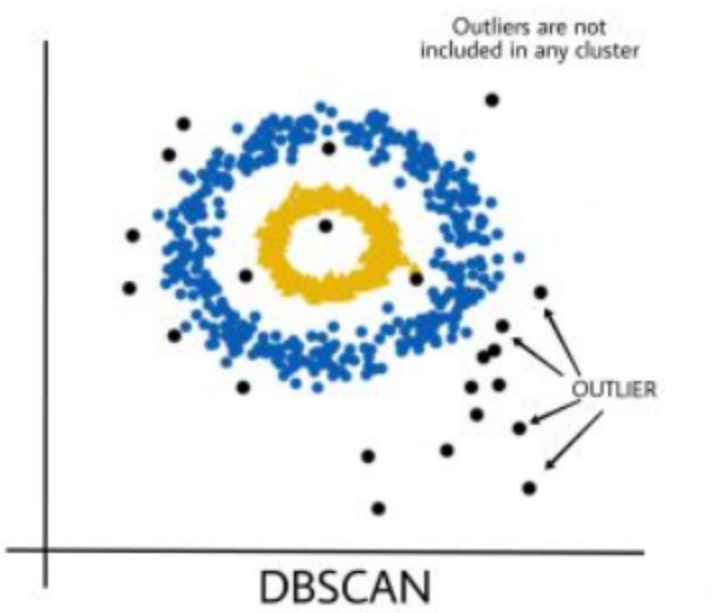
identificar potenciales valores óptimos de *K* (*elbow*, *shilouette*), pero todas ellas son orientativas.

- Dificultad para detectar *clusters* alargados o con formas irregulares.
- Las agrupaciones resultantes pueden variar dependiendo de la asignación aleatoria inicial de los centroides. Para minimizar este problema, se recomienda repetir el proceso de *clustering* entre 25-50 veces y seleccionar como resultado definitivo el que tenga menor suma total de varianza interna. Aun así, solo se puede garantizar la reproducibilidad de los resultados si se emplean semillas.
- Presenta problemas de robustez frente a *outliers*. La única solución es excluirlos o recurrir a otros métodos de *clustering* más robustos como *K-medoids (PAM)*.

2.4 - DBSCAN

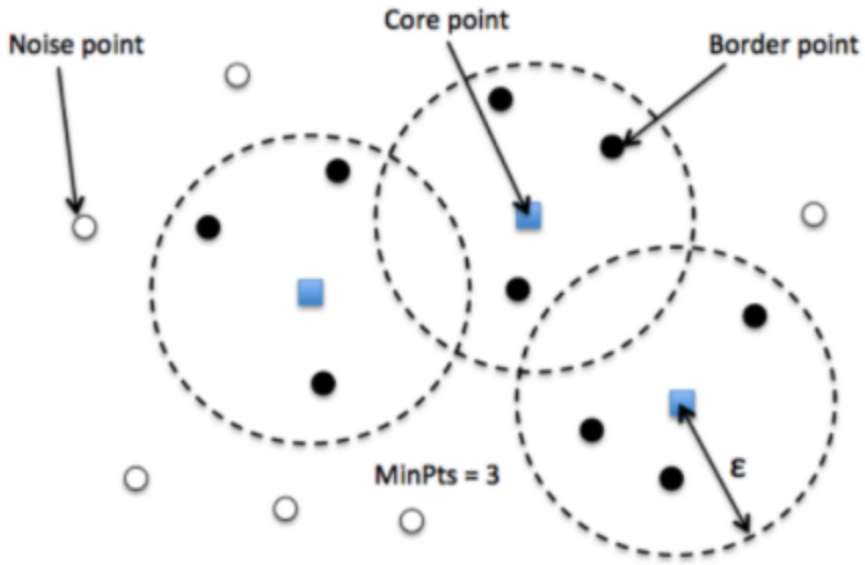
Density-Based Spatial Clustering of Applications with Noise (DBSCAN) es otra técnica de Clustering, un poco menos utilizada que k-means, pero igualmente importante. En esta técnica no es necesario seleccionar la cantidad de clusters, ya que el modelo los definirá automáticamente a partir de la densidad de puntos y no de centroides, como k-means. También incorpora la dimensión de outliers (los llama noise), los cuales deja sin clasificar.

El modelo permite agrupar formas más complejas, no necesariamente globulares como es el caso de k-means.



Los clusters son regiones densas en el espacio de datos. La idea central es que cada punto del cluster tiene que tener un mínimo de vecinos en un radio determinado para no ser un outlier (noise). Entonces, una instancia puede ser noise (ruido) o pertenecer a un cluster. Pero hay tres tipos de puntos, Core Point, Border Point y Noise o Outlier:

- **Core Point:** si tiene igual o más MinPts que los establecidos (3 en el caso de la imagen).
- **Border Point:** si tiene menor MinPts que los establecidos pero está en el radio de un Core Point.
- **Noise o outlier:** si tiene menor MinPts que los establecidos y no está en el radio de un Core Point.



Los dos conceptos (e hiperparámetros) claves en la implementación de este algoritmo son Epsilon y MinPoints:

- **Epsilon (eps)**: define la magnitud del radio que considera como vecinos a los puntos cercanos. Si la distancia de dos puntos es menor o igual al eps, son considerados vecinos.
- **MinPoints (MinPts)**: define la cantidad mínima de vecinos para considerar a un punto parte del cluster.

¿Cómo es el proceso de entrenamiento?

Dados los valores para Epsilon y MinPoints, se realiza el siguiente proceso sobre todos los puntos del dataset:

1. Se toma un punto no visitado random. Se identifica si el punto es un *core*, es decir, si tiene MinPoints en su vecindario. Si no tiene, se lo llama *noise*. Este punto se marca como visitado.
2. Si es un *core*, se le asigna un nuevo cluster y todos los puntos de su vecindario se consideran dentro de su cluster. Si alguno de estos puntos también son cores, este proceso se repite. A los puntos asignados a un cluster que no son core, se los llama *border*. Todos se marcan como visitados.
3. Este proceso se repite hasta que todos los puntos hayan sido visitados.

Si te confunden los pasos, ¡no te preocupes! **Lo importante es que prestes atención a que se incorpora la noción de ruido o outlier, y que no hay que definir a priori la cantidad de clusters que buscamos.**

Aquí te compartimos 4 observaciones para tener en cuenta:

- Cuanto más grande es el dataset, tendemos a seleccionar un MinPts más grande.
- El MinPts debería ser siempre de al menos 3 y tendrá más MinPts a medida que el dataset tenga más dimensiones.
- Si el eps es muy chico, gran parte del dataset va a ser considerado noise.
- Si el eps es muy grande, todos los datos van a estar agrupados por un cluster.

Para Pensar: ¿ves alguna relación entre los últimos dos puntos mencionados y un modelo con overfitting y underfitting?

Visualizar DBSCAN

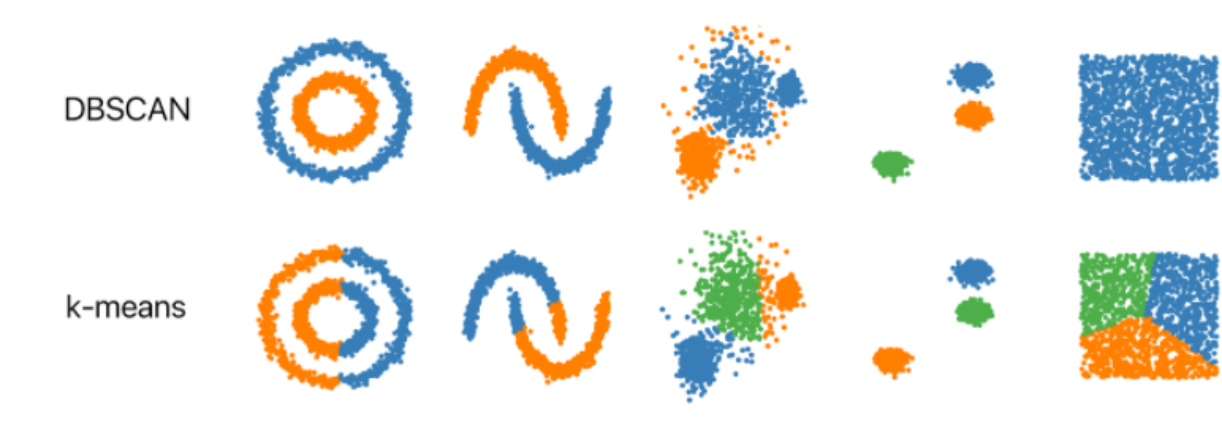
> https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

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

2.5¿Cuándo usar K-means y cuándo DBSCAN?

En primer lugar, es importante mencionar que la mejor práctica para trabajar con clustering -esto también aplica a otros modelos- es probar varios modelos hasta encontrar el que mejor se ajusta a tu problema. Igualmente, eso no hace menos importante comprender las especificidades de cada uno.

Algo que podemos notar rápidamente y con el ojo humano, es que **k-means se adecúa mejor a clusters alejados y bien agrupados y globulares**. Por otro lado, **DBSCAN es más flexible y permite adaptarse a formas más complejas: concéntricas por poner un ejemplo**.



Cómo no todo es tan fácil en la vida -¡o al menos en Data Science!- normalmente tenemos un *trade-off* entre uno y otro. Aquí encontrarás algunos pros y contras de cada uno:

K-Means	DBSCAN
👍 • Rápido, muy mucho. O(n). • No tiene parámetros • Fácil asignar nuevas instancias	• No hay que elegir el número de clusters • Detecta cualquier forma de clusters • Determina automáticamente datos outliers
👎 • Hay que definir el número de clusters • Solo funciona bien con clusters tipo esferas • Sensible a outliers (afectan el promedio)	• Hay que elegir bien los parámetros • No anda bien si hay clusters de diferentes densidades • Es computacionalmente más costoso (tarda más)

2.6 - Otros modelos de Clustering

Existen infinidad de modelos para clustering. En esta bitácora te contamos acerca de dos muy utilizados, pero te recomendamos que explores otros si vas a comenzar un proceso de clusterización real. ¡Es fundamental probar y comparar distintos modelos para este enfoque!

Aquí te mostramos algunos otros modelos utilizados:

- **K-mediods algorithm.** Los centroides son difíciles de definir cuando los atributos son categóricos. Este modelo es una variación de k-means y Mean-Shift, pero que utiliza un centro que no es el promedio de los puntos. Es una buena opción cuando trabajamos con atributos categóricos.
- **Mean-Shift Clustering.** Es un algoritmo similar por su funcionamiento a DBSCAN, pero basado en centroides. Itera entre distintos candidatos a ocupar el centro.
- **Mixturas Gausseanas.** Es un modelo que se puede entender como una extensión de algunas ideas detrás de k-means, pero más poderoso a la hora de estimar clusters complejos.

Existen muchos más, pero no te abrumes por la cantidad de técnicas. No es necesario conocerlas todas juntas: ¡la práctica en problemas reales te irá guiando!

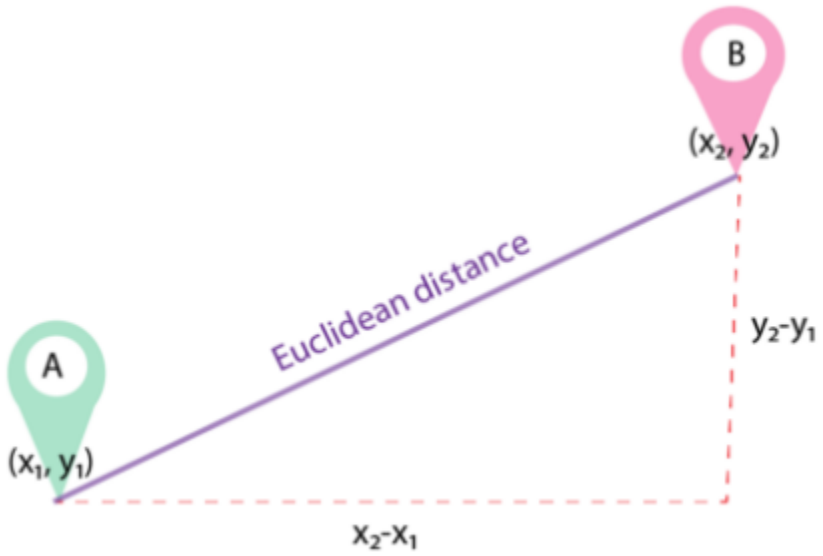
3 - Medidas de Distancia

Todos los métodos de *clustering* tienen una cosa en común, para llevar a cabo las agrupaciones necesitan definir y cuantificar la similitud entre las observaciones. El término distancia se emplea dentro del contexto del *clustering* como cuantificación de la similitud o diferencia entre observaciones. Si se representan las observaciones en un espacio pp dimensional, siendo pp el número de variables asociadas a cada observación, cuando más se asemejen dos observaciones más próximas estarán, de ahí que se emplee el término distancia. La característica que hace del *clustering* un método adaptable a escenarios muy diversos es que puede emplear casi cualquier tipo de distancia, lo que permite al investigador escoger la más adecuada para el estudio en cuestión. A continuación, se describen algunas de las más utilizadas.

3.1 - Distancia Euclídea o Euclídeana

Cuando hablábamos de distancias antes, pensamos principalmente en distancias en una línea más o menos recta.

Si pensamos en distancias entre dos ciudades, pensamos en cuántos kilómetros tenemos que recorrer en una carretera.



Estos ejemplos de distancias que podemos pensar son ejemplos de *distancia euclidiana* . Básicamente, mide la longitud de un segmento que conecta dos puntos. Veamos esto en un gráfico:

Para n puntos la formula general sería:

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

Donde x e y son dos vectores.

La distancia euclidiana es la distancia más utilizada para los algoritmos de aprendizaje automático. Es muy útil cuando nuestros datos son continuos. También se llama L2-Norm .

3.2 - Distancia Coseno

Similitud de coseno y distancia de coseno: Esta métrica es muy utilizada en los sistemas de recomendación. En la métrica de coseno medimos el grado de ángulo entre dos documentos / vectores (el término frecuencias en diferentes documentos recopilados como métricas). Esta métrica en particular se usa cuando no importa la magnitud entre los vectores sino la orientación.

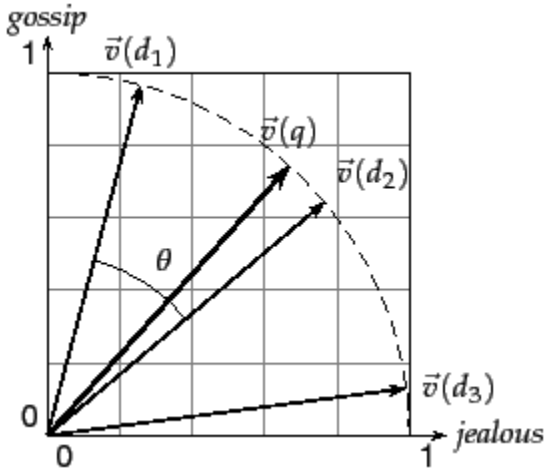
La fórmula de similitud del coseno se puede derivar de la ecuación de los productos escalares:

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\theta$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

La idea básica detrás de la similitud del coseno y la distancia del coseno es que si la distancia del coseno aumenta, la similitud del coseno disminuye y viceversa.

Cosine_distance = 1 - cosine_similarity



Entendamos el principio de funcionamiento del sistema de recomendación sobre la base de la similitud del coseno. Hay dos características a lo largo de los ejes xey. Suponga que necesitamos proporcionar una recomendación para el punto v (d3). El sistema de recomendación calculará la similitud del coseno de acuerdo con todos los demás puntos presentes en la gráfica y el punto que tiene el coseno máximo. Se recomendará similitud (en este caso v (d2)).

Por tanto, es un juicio de orientación y no de magnitud: dos vectores con la misma orientación tienen una similitud de coseno de 1, dos vectores a 90 ° tienen una similitud

de 0, y dos vectores diametralmente opuestos tienen una similitud de -1, independientemente de su magnitud.

La similitud de coseno se usa particularmente en el espacio positivo, donde el resultado está claramente delimitado en [0,1]. Una de las razones de la popularidad de la similitud de coseno es que es muy eficiente de evaluar, especialmente para vectores dispersos.

4 - Evaluación de Clusters

Como aprendieron la clase pasada, la evaluación de los modelos es una parte fundamental en un proceso de Data Science. Necesitamos métricas que nos indiquen si nuestro modelo está ajustando correctamente al problema que estamos trabajando.

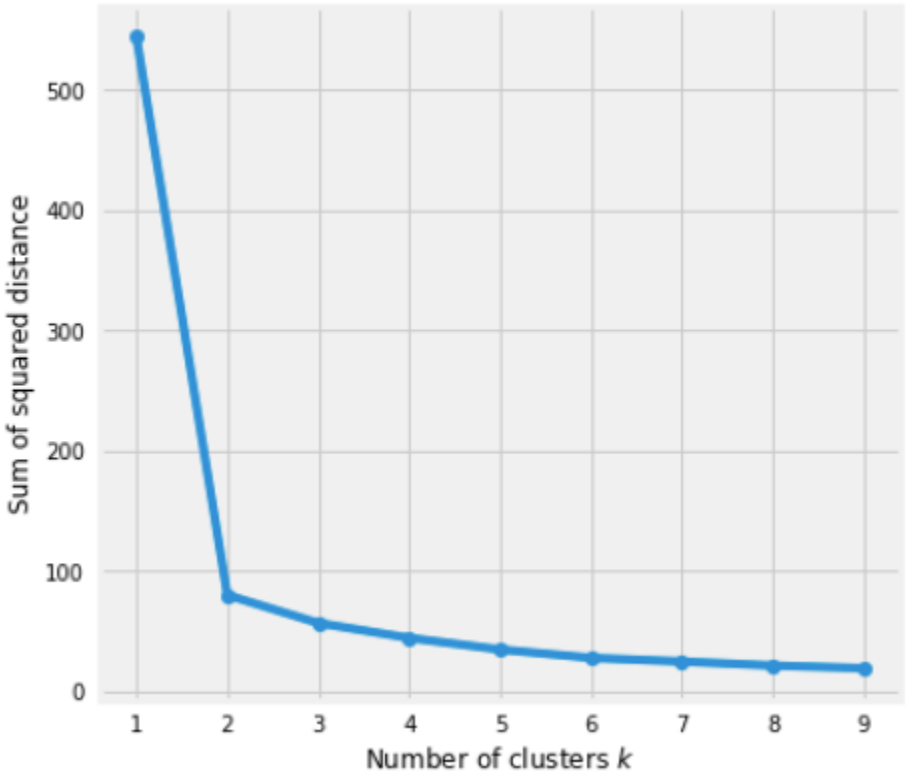
Todos los entrenamientos que realizamos formaban parte del universo del Aprendizaje supervisado. En ese paradigma los datos de entrenamiento tienen una etiqueta con la cual podemos comparar nuestras estimaciones, conocer el margen de error y realizar ajustes. De ahí surge la exactitud, una de las métricas más utilizadas en Data Science, y muchas otras más.

En el **Aprendizaje no supervisado,** y en particular en **Clustering,** no contamos con etiquetas *a priori*. De hecho, ni siquiera sabemos si la estructura de clusters que encontramos es apropiada para nuestro dataset. Por suerte existen algunas técnicas que te darán información importante sobre las particiones obtenidas. ¿Se te ocurren algunas primeras ideas de cómo podés evaluar modelos de clusterización?

4.1 - Elbow method

El método Elbow parte de la premisa de que los centroides en KMeans deben estar lo mas cercanos y comprimidos posible, con el fin de encontrar la mejor cantidad K de clusters que ajustan a nuestro problema. Para esto **mide la distancia de cada punto a su centroide.**

En este gráfico podemos observar el compromiso entre cantidades de clusters y la distancia media de los puntos a su centroide para un dataset:



La tendencia de este gráfico, en el que hay una caída brusca y posteriormente una baja sostenida, es usual en la metodología K-means. Este efecto es razonable cuando reflexionamos un poco sobre la metodología: en un extremo, cuando la cantidad de clusters es igual a la cantidad de instancias en el dataset, tendríamos 1 cluster por punto y la distancia sería 0 para todos. En el otro extremo tendríamos 1 cluster que abarque todos los puntos. Entonces, a medida que aumentemos el número de K, el modelo va a tender al overfitting.

¿Cuál es el número óptimo de clusters? El número óptimo se encuentra cuando la curva detiene la caída abrupta y empieza a disminuir lentamente. Con un poco de imaginación, la curva tiene forma de brazo flexionado. De ahí el nombre del método - en español, codo - para obtener el valor óptimo de K, que suele encontrarse en la punta del codo, en este caso en k=2. En las ocasiones que la caída es constante, puede ser difícil definir este hiperparámetro.

Este método permite medir la distancia de los puntos a sus centroides de dos formas distintas:

- **Inercia:** la suma de la distancia al cuadrado de cada punto con su respectivo centroide. En Sklearn el método “model.inertia_” te facilita esta métrica. Se utilizó esta variable para el gráfico que mostramos previamente.
- **Distorsión:** el promedio de todas las distancias de los centroides con sus respectivos puntos al cuadrado.

¿Cómo es la implementación?

1. Importamos las librerías y definimos la distancia que vamos a utilizar.
2. Realizamos un *for* que entrene modelos iterando entre un array con distintos valores para el hiperparámetro K. No te olvides de guardar los resultados de cada distancia según cada K.
3. Graficamos.

4.2 - Silhouette Score

Una característica de un buen agrupamiento de puntos es la separación entre cada cluster. El ***Silhouette Score*** nos dice exactamente eso, ya que analiza si cada nodo está más cerca de su comunidad que del resto.

Idea: medimos qué tan parecidos son los los datos con su propio cluster (cohesión) en comparacióncon qué tan parecidos son a otros clusters (separación).

En primer lugar, consideramos:

- La distancia *a(i),* que es la distancia promedio entre la punto *i* y el resto de los puntos de su cluster. Notar que, cuanto más chica, mejor, ya que significa que el punto está cerca de su cluster. Pero por sí sola no es fácil determinar si es chica o no, ya que necesitamos algún nivel de referencia.
- La distancia *b(i),* que es la distancia de ese punto al cluster más cercano. Existen varias formas de medir esa distancia, en particular se suele utilizar la distancia promedio entre esa instancia y el cluster más cercano (que no es al que pertenece). Esta distancia, cuanto más grande, mejor. Pero nuevamente no sabemos qué grande o chico sin una referencia.

Lo que podemos hacer es comparar ***a(i)***contra ***b(i)***. Recuerda que queremos el primero sea chico, y el segundo sea grande. Entonces, el coeficiente ***a(i)***/***b(i)*** puede ser una buena medida: a medida que a(i) es más chico, y b(i) es más grande, su división se hace chica, tendiendo a cero. El *Silhoutte Score* parte de esta idea y genera un coeficiente para cada punto, indicando si el punto está más cerca de su cluster, o del cluster más lejano.

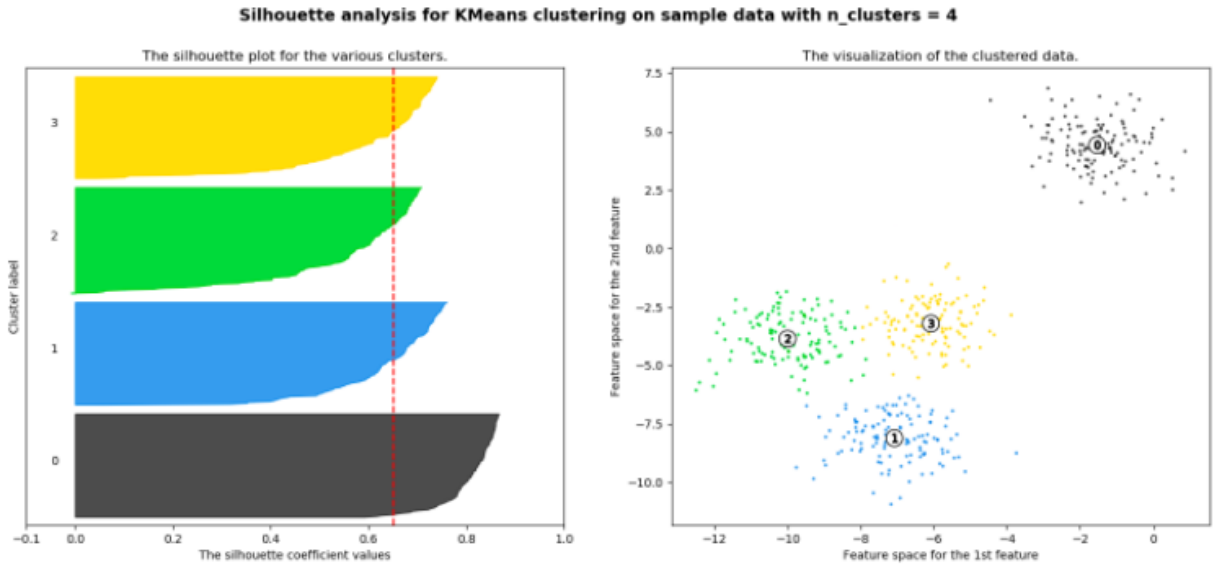
Fórmula: Esto nos da una medida para cada dato de qué tan bien esta ubicado en los clusters. Para una medida de todo el conjunto, se toma la media de todos los S(i)

Insistimos: tómate unos minutos para entender la fórmula. Verás que es menos complejo de lo que parece.

Para Pensar: presta atención a que el coeficiente de Silhoutte va de -1 a 1. ¿Cuándo vale 1 (o cerca de 1)? ¿Cuándo vale 0 (o cerca de 0)? ¿Y -1?

A partir del Silhoutte score para cada instancia/punto del dataset, se suele calcular el Silhoutte promedio de la partición. Así como es el caso para cada score individual, el Silhouette promedio también va del -1 al 1, donde en 1 los clusters están bien separados (es lo que solemos buscar), en 0 están cerca y en -1 el los clusters están mezclados. Esto nos da una idea de cuán buena es la separación en clusters. Notar que estamos diciendo que la separación es buena - o mala - sin usar información externa - por ejemplo, etiquetas -, solamente la partición obtenida y su distribución en el espacio (topología).

Otra cosa que se suele hacer con el Silhouette Score de cada punto es plotear los resultados de la siguiente manera:



Para entender mejor el gráfico:

1. Fíjate que en el eje *x* está el valor de Silhouette, en el eje *y* los clusters.
2. El valor de silhouette de cada instancia está graficado como una barrita muy finita. Están todas las instancias graficadas, ordenadas de mayor a menor.
3. La línea punteada roja representa el Silhouette promedio de toda la partición.

Para Pensar: ¿Te imaginas por qué los valores disminuyen en la base del gráfico de Silhouette de cada cluster? ¿A qué instancias corresponden?

En este ejemplo los clusters se ven bien separados, y de hecho ese es el típico gráfico buscado. Todos los coeficientes están cerca del Silhouette promedio, algunos por arriba, otros por abajo, pero no hay grandes discrepancias.

Te animamos a que entres a la fuente de este gráfico donde podrás ver distintas variaciones con distintas cantidades de clusters. Además, es la documentación original de Scikit-Learn, siempre muy recomendable y útil para cuando lo tengas que implementar. Pero en el encuentro retomaremos estos ejemplos.

Contra: la dificultad que podemos encontrar en este método es su costo computacional. SI bien anda bien y rápido con los datos que solemos trabajar en la carrera, a medida que trabajes con bases de datos más grandes esto se puede volver una complicación. Esto se da porque aplica el cálculo sobre cada punto del dataset.

5 - Hands-On