Práctica 3: Patrones de diseño

Introducción

Como ya conoces de teoría, los Patrones de diseño son soluciones bien conocidas a problemas que aparecen frecuentemente en programación.

Sesiones

Esta práctica la vas a desarrollar en 2 sesiones.

Objetivos

En esta práctica vas a implementar dos patrones de diseño para mejorar tu código, desde el punto de vista de la arquitectura de la aplicación.

Los Patrones de diseño que vas a implementar son:

1. Estrategia (Strategy).
2. Método Fábrica o factoría (Factory method).

Patrón de diseño Estrategia (Strategy)

Introducción

Recuerda la definición del patrón de diseño Estrategia:

> El patrón de diseño Estrategia define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usen.

Por otro lado, recuerda que has utilizado la distancia euclidiana, cuando implementaste el algoritmo KNN, y también en la implementación del algoritmo k-means, para calcular la **distancia** entre las muestras de tu base de datos, o entre nuevas muestras y las muestras existentes en tu base de datos.

La distancia euclidiana es una de las muchas definiciones de distancia que existen. Otra distancia que has utilizado, quizás sin ponerle nombre, es la distancia L_1, distancia de Manhattan (https://en.wikipedia.org/wiki/Taxicab_geometry) o distancia del taxista. Las distancias con esta métrica se miden como diferencias entre las coordenadas del punto origen y el punto destino. La idea es que, si vivo en una ciudad cuadriculada, para ir de un punto a otro de ella, tengo que andar por calles en sentido norte-sur y este-oeste, cualquier otra dirección no es posible. Su definición es la siguiente:

$$d_1(p, q) = \|p - q\|_1 = \sum_{i=1}^{N} |p_i - q_i|$$

Si la comparas con la distancia euclidiana:

$$d_2(p, q) = \|p - q\|_2 = \left(\sum_{i=1}^{N} |p_i - q_i|^2 \right)^{1/2}$$

verás que la diferencia está en la potencia de la función norma ($||\cdot||$) y en la raíz que se extrae del sumatorio. El resultado en ambos casos es un número real.

Resumiendo, tenemos varias alternativas para resolver el mismo problema: calcular la distancia entre dos puntos. Por lo que, el patrón de diseño Estrategia nos viene como anillo al dedo: la familia de algoritmos la forman las distintas definiciones de distancia. Una vez implementado este patrón, podrás probar cómo se comporta un algoritmo de Aprendizaje Automático, sobre el mismo conjunto de datos, al cambiar el modo con el que se calcula la distancia. De manera concreta, puede que, sobre el cojunto de datos *iris* el algoritmo **KNN** clasifique de modo correcto nuevas muestras tan sólo con cambiar la manera de calcular la distancia entre ellas. De igual modo, puede ocurrir que **Kmeans** agrupe de manera distinta un conjunto de muestras tan sólo con cambiar la manera de calcular la distancia entre las muestras.

Metodología

Para incorporar el patrón de diseño Estrategia vas a seguir los siguientes pasos:

1. Crea una interface y nómbrala **Distance**. Esta interface contendrá únicamente el método **double calculateDistance(List<Double> p, List<Double> q)**.
2. Extrae el método en el que calculas la distancia euclidiana y encapsúlalo en una nueva clase a la que nombres como **EuclideanDistance**, que debe implementar la **interface Distance**.
3. Crea una nueva clase que implemente la **interface Distance**, nómbrala **ManhattanDistance**, esta clase debe implementar la distancia de Manhattan.

Ahora, para que las clases **KNN** y **Kmeans** puedan utilizar las clases que implementan el cálculo de la distancia:

1. Añade en las clases **KNN** y **Kmeans** un nuevo atributo **Distance distance**. En este atributo es donde inyectaremos el objeto que implemente el cálculo de distancia que nos interese.
2. Modifica la lista de argumento en los constructores de las clase **KNN** y **Kmeans** para que reciban un nuevo argumento de tipo **interface Distance**.
3. Modifica el método que calcula la distancia en las clase **KNN** y **Kmeans** para que deleguen el cálculo en la referenica **Distance distance**.

Por último, aunque no forma parte del patrón, resulta interesante garantizar que las clases **KNN** y **Kmeans** disponen de un método con el cuál «inyectar» la implementación de la **interface Distance** que nos interese, para ello:

1. Define la nueva **interface DistanceClient**, con un único método **void setDistance(Distance distance)**.
2. Modifica las clase **KNN** y **Kmeans** para que implementen la nueva **interface DistanceClient**.

Pruebas

Prueba, al menos, el siguiente caso:

1. En general, **Kmeans** agrupa de modo distinto las muestras en el conjunto de datos *iris* si utilizamos la distancia euclideana o la distancia de Manhattan.

Patrón de diseño Fábrica (Factory method)

Introducción

El patrón de diseño Factory Method te permite encapsular la creación de instancias de clases que implementen la misma interfaz. Su definición es:

> El patrón de diseño Factory Method define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

En nuestro caso, vamos a crear una fábrica para obtener instancias de clases que implementen la **interface Distance**.

Metodología

Vas a implementar la fábrica de modo parametrizado:

1. Define la **enum DistanceType** con dos elementos: EUCLIDEAN, y MANHATTAN.
2. Define la **interface Factory** con el método **Distance getDistance(DistanceType distanceType)**.
3. Define la **class DistanceFactory** que implemente la **interface Factory** y defina de manera adecuada el método declarado en la **interface Factory**.

Pruebas

Prueba, al menos, los siguientes casos sobre las fábricas:

1. Piden a la fábrica de distancias una instancia de distancia Euclidea y comprueba que efectivamente se calcula la distancia con esta métrica.

Seguir aprendiendo

En esta sección, lo que se propone son ejercicios que puedes realizar por tu cuenta si el tema de Machine Learning te está gustando.

Ten en cuenta que lo que aquí se te propone no supone ninguna mejora de la nota, es, símplemente, alguna idea para que sigas aprendiendo sobre el tema.

Puedes probar a implementar otras medidas de distancia. En este artículo (https://pdodds.w3.uvm.edu/research/papers/others/everything/cha2007a.pdf).

Una vez que hayas implementado alguna distancia más, puede probar cómo agrupa **Kmeans** las muestras del conjunto de datos *iris*.

También, puedes probar qué tal funciona el algoritmo **KNN** cuando usas distintas distancias. Para ello, usa sólo una parte de los datos de *iris* para entrenar el modelo, por ejemplo el 80% de los datos de cada clase, y usa el 20% de los datos restantes para estimar la clase a la que pertenecen. Compara cómo se comporta el mismo algoritmo **KNN** con sólo cambiar la métrica para calcular distancias.