Práctica 2: Programación orientada a objetos. Tipos genéricos y excepciones

Introducción

En esta práctica vas a mejorar tu código gracias a la introducción de una interfaz genérica. Además, vas a incluir excepciones que describan situaciones anómalas durante la ejecución del código.

Sesiones

Esta práctica la vas a desarrollar en 2 sesiones.

Objetivos

En esta segunda prácticas vas a:

1. Refactorizar tu código para hacer uso de una interfaz genérica.
2. Implementar el algoritmo de agrupamiento k-means, para agrupar los datos de la base de datos iris.
3. Definir excepciones para lanzarlas al detectar situaciones anómalas.

Refactoriza tu código para usar una interfaz genérica

En la práctica pasada implementaste dos algoritmos de Aprendizaje Automático: Regresión Lineal y KNN.

Te pedimos que definieses los siguientes métodos en el algoritmo de regresión lineal:

```java
public void train(Table data); // Lee los datos de un fichero y construye el modelo.
public Double estimate(Double sample); // Devuelve una estimación para el «sample»
```

y estos métodos para el algoritmo KNN:

```java
public void train(TableWithLabels data); // Lee los datos de un fichero y construye el modelo.
public String estimate(List<Double> sample) // Devuelve una estimación para el «sample»
```

Fíjate que para ambas clases, el cometido de los siguientes métodos es el mismo:

- **train** construye el modelo a partir de un conjunto de datos.
- **estimate** da una estimación para una nueva muestra.

Para el caso del método **train** la diferencia es clara, en el algoritmo de regresión lineal pasamos como argumento del método una **Table** y en el algoritmo KNN pasamos como argumento una **TableWithLabels**. Por otro lado, entre ambas clases del argumento existe una relación madre (**Table**) a hija (**TableWithLabels**).

En el caso del método **estimate** hay dos claras diferencias entre las dos versiones del método:

1. El tipo de retorno es distinto: **Double** para la regresión lineal, y **String** para KNN.
2. El tipo del argumento es distinto: **Double** para la regresión lineal, y **List<Doule>** para KNN.

En el caso del método **estimate** no tenemos tanta suerte como con **train**, ya que los tipos no esta relacionados entre sí por herencia.

¿Cómo vamos a proceder?

1. Ya que el cometido de los métodos con el mismo nombre en clases distintas es el mismo, vamos a intentar unificarlos en una **interface**.
2. Para poder trabajar con distintos tipos, tanto en los argumentos como en el valor de retorno, vamos a trabajar con tipos genéricos.

Metodología

1. Crea una interfaz **Algorithm** con un método **train** que use genéricos para el argumento, y que además garantice que ese argumento es **Table** o una clase hija de **Table**, como por ejemplo **TableWithLabels**.
2. Declar el método **estimate** en la **interface Algorithm** con un tipo genérico para su argumento, y otro tipo genérico para el valor de retorno.
3. Modifica la clase **LinearRegresion** para que implemente la **interface Algorithm** asignando a los genéricos los tipos adecuados en la definición de la clase **LinearRegresion**.
4. Modifica la clase **KNN** para que implemente la **interface Algorithm** asignando a los genéricos los tipos adecuados en la definición de la clase **KNN**.
5. Incorpora a tu código alguna excepción que describa una situación anómala. Por ejemplo, una división por cero en el caso del algoritmo de la regresión lineal; cuando el número de clusters es mayor que el de datos leídos del fichero CSV; cualquier otra que te parezca importante.

> *Refactorizar* significa mejorar el código sin cambiar su funcionalidad. Por ejemplo, si un método realiza más de una tarea, podemos dividirlo en varios métodos pequeños cada uno de los cuales realiza una tarea única del método original.
>
> Como tenemos una batería de pruebas unitarias y usamos git, no debes tener miedo a «romper» tu código en las sesiones de refactorización. Siempre podrás volver a última versión estable en tu repositorio.

Agrupamiento (clustering) con k-means

Introducción

Considera los datos del fichero iris.csv, y supón que no te han proporcionado las etiquetas de cada uno de los ejemplares de flor de Iris que allí aparecen. Ahora te piden que las agrupes, es decir, que crees tres grupos de flores (fíjate que el número de grupos es un parámetro conocido), uno para cada especie de Iris, y que asignes cada flor del grupo inicial a alguno de esos grupos, atendiendo a sus características (la longitud y anchura de pétalos y sépalos). ¿Cómo lo haría?

Existen varias posibilidades pero todas ellas descansan en la misma idea intuitiva: *La distancia entre los ejemplares del mismo grupo es menor que la distancia entre ejemplares de grupos distintos*, aunque esta afirmación no siempre se cumple para todos los ejemplares.

Esta idea también se puede expresar del siguiente modo: *Si existe un representante de cada grupo, todos los elementos de ese grupo están más cerca del representante de su grupo que de los representantes de los otros grupo*.

La idea anterior es la que usa k-means (https://es.wikipedia.org/wiki/K-medias), encontrar representantes para cada grupo

de manera que la distancia entre los ejemplares de una grupo y su representante es más pequeña que la distancia de los ejemplares a los representantes de los otros grupos.

La clase que implemente el algoritmo Kmeans debe implementar la **interface Algoritm** indicando el tipo concreto que va a utilizar.

Metodología

Una vez que hayas leído los datos de un fichero CSV, los pasos que debe implementar el método **train(Table t)** son:

1. Elige de los puntos que has leído del CSV, aleatoriamente, un representante para cada grupo. Comprueba que no tengas ningún punto repetido.
2. Calcula la distancia de cada elemento a cada uno de los representantes de cada grupo.
3. Asigna a cada elemento el grupo del representante al que está más cercano.
4. Una vez asignados los grupos a cada elemento, calcula el centroide de cada grupo, y tómalo como nuevo representante del grupo.
5. Vuelve a 3 y repite un número fijo de veces.

En el primer paso de este algoritmo has hecho una elección aleatoria. Es conveniente que puedas fijar la **semilla** de generación de los número aleatorios para que los resultados sean los mismos siempre que utilices la misma semilla. Para fijar la semilla puedes utilizar **Random random = Random(long semilla)**, y cuando necesites generar un valor aleatorio.

El constructor de la clase Kmeans será algo como:

```java
public Kmeans(int numberClusters, int iterations, long seed)
```

donde indicarás en **int numberClusters** el número de grupos, en **int iterations** el número de iteraciones, y en **long seed** la semilla para generar números aleatorios.

Además, el método genérico, que ahora es genérico, **tipoRetorno estimate(muestra)** debe devolver un String que sea la etiqueta del cluster al que pertenece la muestra, por ejemplo «cluster-1» si pertenece al primer cluster, «cluster-2» si pertenece al segundo, etcétera.

El centroide de un conjunto de puntos es el punto cuyas coordenadas se calculan como el promedio del valor de las coordenadas del conjunto de puntos, que para puntos con tres dimensiones (x, y, z) se escribe como:

$$\vec{C}(x,y,z) = \frac{\sum_{i=1}^{M} \vec{e_i}(x,y,z)}{M} = \frac{1}{M}\left(\sum_{i=1}^{M} x_i, \sum_{i=1}^{M} y_i, \sum_{i=1}^{M} z_i\right)$$

Es decir, cada una de las coordenadas del centroide es la media de las coordenadas del conjunto de puntos.

El número de agrupamientos es un parámetro de entrada del algoritmo. Por simplicidad, el número de iteraciones también será un parámetro de entrada del algoritmo.

Idealmente, el último paso del algoritmo no es repetir los pasos 3-5 un número fijo de veces, sino hasta que el algoritmo converja. La convergencia no siempre está garantizada, por lo que simplificaremos el algoritmo para que resulte más sencilla su implementación.

Finalmente, debes implementar el método **estimate(Row r)** de tal modo que al proporcionarle un nuevo ejemplar **Row r**, devuelva el grupo al que el algoritmo estima que pertenece ese ejemplar. Si nos damos cuenta que realmente, (el método **train** de) Kmeans nos proporciona agrupamientos con etiqueta, podemos utilizar cualquier algoritmo de clasificación de Aprendizaje Automático para hacer un estimate desde un conjunto de datos etiquetados (como por ejemplo, el k-vecinos más próximos - KNN - que hemos implementado en la práctica anterior). Sin embargo, para esta práctica, es suficiente asignar a la muestra la etiqueta del cetroide más cercano.

Pruebas

Tal y como lo hemos implementado, el algoritmo k-means no es determinista, es decir, en cada ejecución puede dar resultados distintos. Esto se debe a que estamos iniciando al azar los primeros centroides. Existen opciones deterministas pero no las vamos a implementar.

Para realizar las pruebas, puedes iniciar el algoritmo con centroides concretos, de los que sepas cual será el resultados final del agrupamiento.

Seguir aprendiendo

En esta sección, lo que se propone son ejercicios que puedes realizar por tu cuenta si el tema de Machine Learning y programción orientada a objectos 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.

Variar el algoritmo **estimate** en kMeans

Como ya hemos mencionado, el método **estimate** en la clase kMeans podría utilizar cualquier algoritmo de Aprendizaje Automático, como por ejemplo, el k-vecinos más próximos (KNN). ¿Puedes hacer que tu método de estimación de kMeans funcione con KNN? Piensa en cómo utilizar los principios de programación orientada a objetos para llegar a una buena solución. ¿Y si deseas inyectar dinámicamente diferentes funciones de estimación (piensa en patrones de diseño)?