Code Review

El siguiente es un reporte de feedback al tp realizado por Federico Amura y Sebastian Gavrilov. Se encuentra en https://github.com/FedericoAmura/Funcional2019C2

Informe

Carga de datos: dbFiller

Me parece muy claro como se ve en el código (en el Main) los dos pasos propuestos para esta primer solución: primero leer las filas del archivo csv y luego insertarlas en la base.

Muy interesante cómo usaron las mónadas. A mi no me quedaba muy claro cómo se utilizaban, en qué casos y que significaban y gracias a ver cómo la usaron ustedes pude entenderlas un poco más. Sino me equivoco al usar la mónada IO lo que están haciendo es encapsular las funciones que interactúan con el exterior dentro de una monada para que si algo falla puedan devolver un valor dentro de la mónada que lo indique. Por ejemplo, si el archivo no existe entonces la función “fromPath” podría devolver algo del estilo IO[Nothing].

No me queda claro porque en el object “DB” tienen como atributos las variables cs y xa, ¿no estarían como guardando estado de esta forma?

Una mejora que propongo, que también propuse para mi tp, sería que para la clase Row cada parámetro sea una case class, esto permitiría realizar validaciones sobre cada columna del csv, como por ej que las columnas dolar_bn y dolar_itau tengan valores mayores o iguales a 0.

Predicción de valores: trainer

Al igual que en el módulo anterior me parece muy claro en el código los pasos que siguieron para resolver lo pedido y como utilizaron las monadas para las funciones que tienen side effects.

En “SparkRFPipeline” les quedo muy claro cómo van procesando los datos para llegar a la forma que los necesita sparkML.

Quizás para el objeto “SparkRFPipeline” las distintas case class que utilizan para representar los distintos tipos de datos que utiliza sparkML podrían estar definidas en otro archivo e importarlas, para que asi se pueda reutilizar ese objeto con otro set de datos distinto.

API de consultas

En este caso no me quedó muy claro en qué momento chequean si los datos pasados a la api ya existen en la base o no. Siguiendo el flujo de ejecución en SoyApp “processRequest()” llama a RandomForest sin preguntar primero si los datos ya existen en la base.

Una vez más me parece muy bueno como utilizaron la monada Try para manejar los casos que podían fallar, lo que me ayudó a entender más cómo utilizar mónadas.