Práctica del Módulo 2: Apache Spark

Nombre del Estudiante: Alex Herrerías Ramírez

11 de enero de 2026

Índice

1. Opinión Personal

Antes de empezar con el ejercicio 2 de la práctica, quería aportar mi feedback sobre el curso de DataCamp, si bien es cierto que es un curso muy completo y junto al ejercicio opcional ha hecho que el desarrollo de esta practica sea muy sencillo lo cual deja bastante claro que el curso si tiene una aplicación real, tal vez por juntarse con las fiestas o por la sensación de repetición se me ha hecho demasiado extenso para el periodo de tiempo/ ser una sola práctica.

Quitando problemas de tiempo el tanto el curso como la plataforma (la cual usaba con anterioridad) son muy recomendables.

2. Introducción y definición del proyecto

2.1. Entorno de desarrollo

Para la realización de esta práctica, y para poder garantizar la reproducibilidad y el aislamiento de las dependencias de mi ordenador, se ha optado por desplegar un entorno local utilizando **Docker Desktop** como se nos recomienda en la asignatura.

Se ha seleccionado la imagen oficial `jupyter/pyspark-notebook` del repositorio de Jupyter Docker Stacks. Esta imagen proporciona un entorno con todas las dependencias necesarias y la interfaz de Jupyter Lab ya configurada lista para ejecutarse en modo local con Spark.

Esta configuración ha permitido simular un nodo de procesamiento Spark asignando recursos específicos más potentes para poder procesar el dataset correctamente con unos hiperparámetros superiores a los que se podría sin estos recursos (memoria y CPU) directamente desde la configuración de Docker.

2.2. Objetivos y selección del dataset

Para el desarrollo del Ejercicio 2, se ha seleccionado el dataset **Rossmann Store Sales** disponible en Kaggle. La elección de este conjunto de datos se fundamenta en la similitud con lo aprendido en DataCamp junto a cumplir con los criterios de evaluación de la asignatura:

1. **Dominio de aplicación real:** Pertenece al sector del *Retail*, un área donde el Big Data tiene una aplicación directa y de alto valor comercial.
2. **Tipología del problema:** Se trata de un problema de **Regresión** parecido al ejercicio opcional de DataCamp, consistente en predecir la variable `Sales` basándonos en datos históricos.
3. **Complejidad:** Permite el uso de un *Pipeline* completo de Spark ML, similar al estudiado en el ejercicio opcional de DataCamp, pero con la dificultad añadida de gestionar series temporales y características categóricas complejas.

El objetivo principal es construir un modelo predictivo capaz de estimar las ventas diarias de las tiendas Rossmann, integrando información sobre promociones, competidores y festividades.

3. Ingeniería de datos con PySpark

3.1. Carga e integración de fuentes

El proyecto parte de dos fuentes de datos CSV: `train.csv` (histórico de ventas) y `store.csv` (metadatos de cada tienda). Se utilizó `spark.read.csv` con la inferencia de esquemas activada (`inferSchema=True`) para una detección inicial de tipos.

Dado que la información relevante para el modelo se encuentra distribuida en ambas tablas, se realizó una operación de `join` (tipo *left*) utilizando el identificador `Store` como clave. Previamente, se aplicó un filtrado para eliminar ruido del dataset de entrenamiento: se descartaron los registros donde las tiendas estaban cerradas (`Open == 0`) o donde las ventas fueron cero, ya que intentar predecir ventas en días de cierre no aporta valor al modelo y distorsiona el aprendizaje.

3.2. Estrategia de tratamiento de nulos

Uno de los retos principales de este dataset fue la gestión de valores nulos en columnas críticas. En lugar de eliminar estos registros, se optó por una estrategia de imputación basada en el conocimiento del dominio:

- **CompetitionDistance:** Los valores nulos en esta variable se interpretaron como la inexistencia de competencia cercana. Por tanto, se imputaron con un valor artificialmente alto (el doble de la distancia máxima observada en el dataset). De esta forma, el modelo interpreta que la influencia de la competencia es despreciable para esas tiendas.
- **CompetitionOpenSince [Month/Year]:** Al no existir datos de apertura de la competencia, se imputaron con 0. Esto permite que el algoritmo trate estos casos como una categoría distinta.
- **Promo2Since [Week/Year] y PromoInterval:** La ausencia de datos aquí indica que la tienda **no participa** en la promoción continua (”Promo2”). Se rellenaron con 0 y ”Nonerespectivamente, permitiendo que el `StringIndexer` posterior procese ”Noneçomo una categoría válida que representa ”Sin Promoción”.

4. Análisis exploratorio de datos (EDA)

Antes de proceder al modelado, se realizó un análisis visual convirtiendo una muestra agregada de los datos a Pandas para utilizar las librerías `Seaborn` y `Matplotlib`. Este paso fue importante para entender la naturaleza temporal de las ventas.

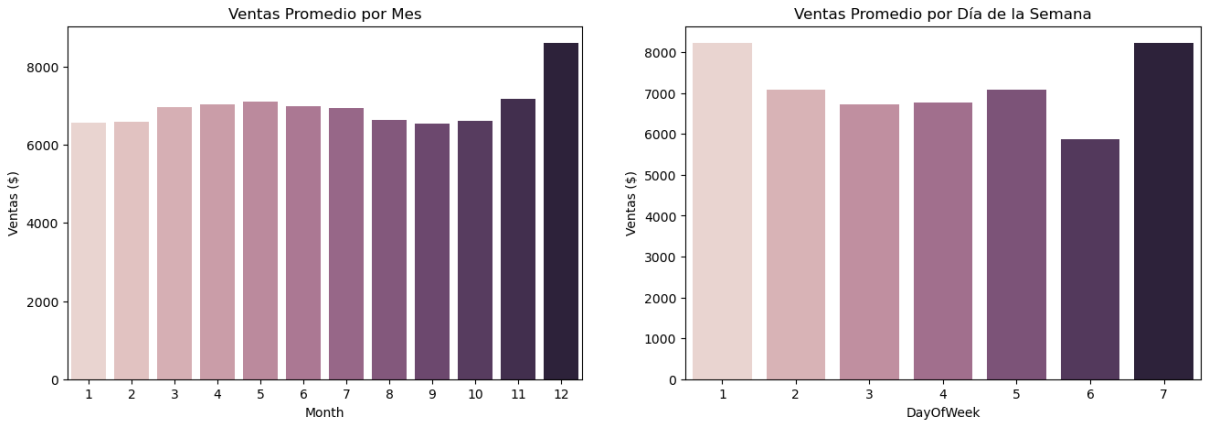

Figura 1: Análisis de ventas promedio: estacionalidad mensual y ciclo semanal.

4.1. Patrones estacionales y semanales

El análisis reveló dos comportamientos que condicionaron la elección de las variables predictoras:

1. **Estacionalidad anual:** Se observó un incremento en el volumen de ventas durante los meses de noviembre y diciembre, correlacionado con la campaña navideña. Esto justifica la necesidad de descomponer la fecha en variables como “Mes” y “Semana del año”.

2. **Ciclo semanal:** Existe una fluctuación dependiendo del día de la semana, con picos de ventas habituales los lunes y caídas los domingos. Esto confirma que el día de la semana es un predictor.

La conclusión principal del EDA es que la relación entre las variables temporales y las ventas **no es lineal**, lo que sugiere que modelos basados en árboles de decisión deberían comportarse mejor que una regresión lineal simple.

5. Ingeniería de características y pipeline

Para preparar los datos para los algoritmos de Spark MLlib, se diseñó un *Pipeline* de transformación compuesto por tres etapas clave:

5.1. Descomposición temporal

Dada la naturaleza de serie temporal del problema, la columna `Date` por sí sola no es procesable directamente. Se procedió a extraer características explícitas: `Year`, `Month`, `Day`, y `WeekOfYear`. Estas nuevas columnas permiten al modelo capturar los patrones estacionales detectados en el EDA.

5.2. Codificación de variables categóricas

Las variables categóricas como `StoreType`, `Assortment` y `StateHoliday` no tienen un orden matemático intrínseco. Para tratarlas correctamente se aplicó una doble transformación:

1. **StringIndexer:** Convierte las cadenas de texto en índices numéricos.

2. **OneHotEncoder:** Convierte esos índices en vectores binarios. Esto es importante para evitar que el modelo asuma una relación de orden errónea.

Nota sobre la variable Store: Debido a la alta cardinalidad de la variable `Store` (más de 1000 tiendas distintas), se aplicó únicamente `StringIndexer` sin *OneHotEncoding* para evitar una explosión en la dimensionalidad del vector de características (*curse of dimensionality*), lo que habría incrementado el consumo de memoria y el tiempo de cómputo.

5.3. Vector Assembly

Finalmente, todas las características transformadas y las variables numéricas originales se consolidaron en un único vector de características utilizando `VectorAssembler`, generando la columna `features` lista para el entrenamiento.

6. Modelado y evaluación

6.1. Estrategia de validación (Train/Test Split)

Un aspecto importante en este proyecto es la división de los datos. Al tratarse de una serie temporal, realizar un *random split* habría sido un error, provocando *data leakage*.

Entrenar con datos futuros para predecir el pasado inflaría artificialmente las métricas de precisión.

Por ello, se implementó una **división temporal estricta**:

- **Train Set:** Todos los registros anteriores al 15 de junio de 2015.
- **Test Set:** Registros posteriores a esa fecha (las últimas 6 semanas del dataset).

Esto simula un escenario real de producción donde el modelo se entrena con el pasado para predecir el futuro inmediato.

6.2. Modelos seleccionados

Se entrenaron y compararon dos algoritmos de regresión pertenecientes a la familia de árboles de decisión:

- **Random Forest Regressor:** Seleccionado por su capacidad de paralelización y robustez frente al sobreajuste (*overfitting*). Se configuró con una profundidad máxima (`maxDepth`) de 12 y 100 árboles.
- **Gradient-Boosted Trees (GBT):** Seleccionado por su alta capacidad predictiva. A diferencia del Random Forest, el GBT construye árboles secuencialmente para corregir los errores de los anteriores.

6.3. Análisis de resultados

Tras la evaluación sobre el conjunto de test (datos no vistos por el modelo), se obtuvieron las siguientes métricas:

Modelo	**RMSE**	**MAE**	**R^2**
Random Forest	1057.03	740.24	0.8786
GBT Regressor	**923.32**	**632.13**	**0.9074**

Cuadro 1: Comparativa de métricas de rendimiento en el conjunto de test.

Los resultados son concluyentes: el modelo **Gradient-Boosted Trees** superó al Random Forest en todas las métricas.

- Un **R^2 de 0.9074** indica que el modelo es capaz de explicar casi el 91 % de la variabilidad en las ventas, un resultado sobresaliente para un problema de retail con alta varianza.
- El **MAE (Error Absoluto Medio) de ∼632€** es muy competitivo, considerando que las ventas diarias de estas tiendas oscilan en rangos de miles de euros.

7. Conclusiones y futuras mejoras

Este ejercicio ha permitido consolidar el uso de Spark MLlib en un entorno local dockerizado. La implementación del *Pipeline* y, sobre todo, la correcta gestión de los datos temporales y nulos, han sido los factores claves para el éxito del modelo.

La superioridad del GBT demuestra que el aprendizaje secuencial (*boosting*) es más efectivo para capturar las complejidades del mercado minorista que el promedio de árboles independientes (*bagging*). Como futura línea de mejora, en un entorno con mayores recursos computacionales como un cluster real, se podría implementar una etapa de *Hyperparameter Tuning* utilizando `CrossValidator` para ajustar de forma más fina parámetros como la profundidad de los árboles o la tasa de aprendizaje.