Solución del Proyecto Final - Modulo 4

Yoksan Varela Cambronero
Big Data

20 de julio de 2024

1. Trasfondo y problema a solucionar: Un nuevo desarrollador de video juegos

De acuerdo con el artículo publicado por *Fortune Business Insights* (https://www.fortunebusinessinsights.com/video-game-market-102548), el valor de mercado de los video juegos era de aproximadamente de $250 mil millones para 2022, y se espera un crecimiento a una tasa compuesta anual (CAGR por sus siglas en inglés) del 13.1 %, lo que implica que alcanzará los $665 mil millones para 2030.

Una nueva desarrolla costarricense de video juegos quiere formar parte de este crecimiento de mercado y obtener una parte de ese dinero. Dicha desarrolladora hizo una inversión importante en todo lo referente a infraestructura para el desarrollo de un nuevo juego, pero pero esto produjo un problema: Para poder recuperar su inversión, los cálculos realizados indican que este nuevo viejo juego tiene que vender más de 175 mil copias, por lo tanto, esta empresa necesita entender cuál es la receta que les permita llegar a esa meta tan agresiva.

1.1. Solución propuesta

Se le plantea a dicha empresa crea una sistema de aprendizaje automático que pueda predecir si el nuevo video juego va a llegar a vender la cantidad deseada o, en su defecto, no va a alcanzar las 175 mil copias en ventas.

2. Fuentes de datos

Para hacer este estudio de mercado, se van a utilizar dos conjuntos de datos tomados de Kaggle:

2.1. Primer set: Video Games Dataset

URL: https://www.kaggle.com/datasets/beridzeg45/video-games
Este conjunto de datos cuenta con información de 14035 video juegos. Los atributos que contiene dicho documento es:

1. *Title:* Nombre del video juego.
2. *Release Date:* Fecha del lanzamiento del video juego.
3. *Developer:* Desarrollador del video juego.
4. *Publisher:* Editor del video juego (quien provee el financiamiento del mismo, incluyendo mercadeo y promoción).
5. *Genres:* Tipo a cual pertenece el video juego, por ejemplo: acción, aventura, roles, deportes, etc.
6. *Product Rating:* Es la calificación del producto, en este caso, quienes deberín jugar el juego: apto para todo publico, mayores de cierta edad, adultos, etc.
7. *User Score:* Puntaje de evaluación impuesto por el jugador final.

8. *User Rating Count:* Conteo de veces que el juego recibió un puntaje por parte de un jugador.

9. *Platforms info:* Información de las plataformas donde se juega el juego, pero en forma de diccionario.

2.1.1. Cantidad de valores Nulos y descarte de atributos

Este análisis empieza con entender la cantidad de valores NaN presentes en el conjunto de datos. Los resultados se tabulan a continuación:

Atributo	**Conteo de NaN**
Title	21
Release Date	64
Developer	138
Publisher	138
Genres	21
Product Rating	3050
User Score	2341
User Rating Count	2756
Platforms info	0

Cuadro 1: Conteo de valores NaN en el set de datos Video Game Dataset

Para lidiar con los valores NaN, se decide reemplazar los NaN en con "Not Specified.[en] los siguientes atributos: *Genres, Publisher y Product Rating*, dado que se consideran atributos de interés para el estudio debido a su conocido impacto en las ventas de un video juego. Además, para el atributo *Product Score* se decide reemplazar los NaN por el valor medio del atributo, por la misma razón explicada anteriormente.
Los siguientes atributos fueron descartados por lo siguiente:

1. User Ratings Count: Este conteo carece de información importante, que la vez es redundante con User Score.

2. Platforms Info: Este atributo se puede obtener del otro set de datos que se explicará posteriormente de una forma más fácil.

3. Developer: Dado que el cliente es un desarrollador en sí, no esta interesado en saber quién desarrolló el juego para evitar comparaciones con desarrolladores más establecidos y con mayor capital de operación.

Ya con estos cambios hecho, se procede a descartar las instancias que tienen NaN en Title y Release Date para obtener el set de datos limpios y proceder a analizar de forma estadística. Al final de estos pasos, quedaron 13991 instancias.

2.2. Segundo set: Video Game Sales 2024

URL: https://www.kaggle.com/datasets/asaniczka/video-game-sales-2024
Este es el conjunto de datos principal, ya que cuenta con la información de las ventas actualizada hasta lo que se lleva de este año 2024. Cuenta con información de más de 64 mil video juegos. Los atributos en este set de datos son los siguientes:

1. *img:* Imagen de la portada del video juego.

2. *title:* Nombre del video juego. **Esta va a ser la llave a utilizar para poder unir los dos set de datos.**

3. *console:* La plataforma donde se corre el video juego.

4. *genre:* El tipo de juego, ejemplos: acción, deportes, estrategia, otros.

5. *publisher:* Editor del video juego.

6. *developer:* Desarrollador del video juego.
7. *critic_score:* Puntaje de la crítica.
8. *total_sales:* Ventas totales a nivel mundial. **De los datos de este atributo se va a generar el objetivo predictivo.**
9. *na_sales:* Ventas totales en Japón.
10. *jp_sales:* Ventas totales en Estados Unidos.
11. *pal_sales:* Ventas totales en Europa y África.
12. *other_sales:* Ventas totales en otras regiones de menor volumen.
13. *release_date:* Fecha del lanzamiento del video juego.
14. *last_update:* Fecha de la última actualización de los datos.

2.2.1. Cantidad de valores Nulos y descarte de atributos

De forma similar con el set de datos anterior, empieza con entender la cantidad de valores NaN presentes en el conjunto de datos. Los resultados se tabulan a continuación:

Atributo	**Conteo de NaN**
img	0
title	0
console	0
genre	0
publisher	0
developer	17
critic_score	57338
total_sales	45094
na_sales	51379
jp_sales	57290
pal_sales	51192
other_sales	48888
release_date	7051
last_update	46137

Cuadro 2: Conteo de valores NaN en el set de datos Video Game Sales 2024

Aunque inicialmente se tenían mas de 64 mil instancias, el hecho que 45094 de ellas no tengan la información relacionada con las ventas totales indica que hay que descartar la mayoría de las instancias. No obstante, la cantidad restante es suficiente para el desarrollo de este proyecto.

Con base a esto, se toman las siguientes decisiones para limpiar los datos: Primero, se eliminan los siguientes atributos:

1. img: No se necesita la imagen de la portada para este análisis.
2. Todos los relacionados con ventas que no sea total_sales: Dado que se quiere vender a nivel mundial, la información regional se descarta dando como prioridad la global.
3. critic_score: Es una práctica muy común que se pague por las buenas revisiones de los medios de la crítica, por lo tanto, este atributo podría introducir este ruido en el estudio.
4. release_date: Información presente en el set de datos anterior.
5. last_update: Información que no es relevante en este estudio.

6. developer: Se descarta por la misma razón mencionada anteriormente.

Luego de esto, de desechan todas las instancias con NaN, lo que deja un total de 18922 instancias con 5 atributos. Es importante mencionar en este punto que, aunque este set de datos se queda con 5 atributos y el anterior con 6, la mayoría de estos atributos categóricos como *console (39), publisher (733 en este último set de datos) y genres (19 en este último set de datos)* contienen mucho valores distintos, lo que da pie a un estudio con más de 800 atributos.

3. Entregables y descripción de los archivos

Dentro del archivo comprimido se encuentras los siguientes archivos:

3.1. Folder Datasets

Contiene los archivos CSV con la información usada para el desarrollo de este proyecto, explicados anteriormente. Esos archivos son:

1. **all_video_games_info.csv**
2. **vgchartz_sales_2024.csv**

Estos fueron renombrados con respecto al nombre original para efectos de lectura por la función de carga de Python. Además, el header de los archivos fue removido para poder crear uno propio y evitar *warnings* durante ejecución.

3.2. Folder Library

Contiene diferentes archivos Python para diferentes funciones necesarias para el correcto funcionamiento de esta solución. A continuación de detalla el contenido de cada archivo, y sus pruebas unitarias (si tiene alguna asociada).

3.2.1. args_parser.py

Contiene la función ArgParser, la cual interpreta los parámetros pasados durante el comando de ejecución del archivo *process_etl_YoksanVarela.py*, que en este caso son los dos archivos CSV mencionados anteriormente.

Esta función es critica para la ejecución general del programa, por lo tanto, la ejecución en si del programa es la validación de la función, y es por eso que no cuenta con funciones unitarias.

3.2.2. data_integrity.py

Este archivo contiene las siguientes funciones que tienen como finalidad verificar y corregir la integridad de los datos:

1. **nan_count(dataframe)**: Retorna un Spark Dataframe con la cuenta de NaN o Null por columna del *dataframe* que se pase como entrada.
2. **fill_nan_with_value(dataframe,column,value)**: Retorna un Spark Dataframe con todos los valores NaN o Null sustituidos por el valor de entrada *value* en la columna de entrada *column*, en el *dataframe* de entrada.
3. **fill_nan_with_mean(dataframe,column)**: Similar a la funcion anterior, retorna un Spark Dataframe con todos los valores NaN o Null sustituidos por la media de todos los valores en la columna de entrada *column*, en el *dataframe* de entrada. Nota: solo funciona con columnas del tipo integer, float o double.
4. **date_format(date,formats)**: Funcion para convertir una fecha en un formato particular. Utiliza un arreglo de formatos *formats* de entrada para ver cuales otros formatos de fecha pueden ser convertidos al deseado.

3.2.3. data_transformation.py

Este archivo contiene las siguientes funciones que tienen como finalidad tranformar datos de diferentes formas:

1. **lower_case(dataframe)**: Retorna un Spark Dataframe donde todos los valores de las columnas tipo string del *dataframe* son convertidos a minúsculas. Esta es una practica de programación muy usada para evitar problemas de ejecución o procesamiento provocado por mayúsculas.
2. **dataframe_joiner_title(dataframe1, dataframe2)**: Función para retornar un Spark dataframe que resulta de la unión horizontal de los dos dataframes pasados por entrada usando un valor llave, en este caso, esta llave es el titulo o nombre del video juego. Nota: solo función con un nombre especifico de las columnas a ser usadas como llaves.

3.2.4. postgresql_db.py

Este archivo contiene las siguientes funciones básicas de escritura (*save_in_db(dataframe, db_name)*) y lectura (*read_from_db(db_name, spark_session)*) para bases de datos POSTGRES. Ambas funciones reciben como entrada el nombre de la base de datos *db_name*. La función de escritura recibe también el *dataframe* que va a ser escrito en la base de datos, y la funcion de lectura recibe una sesión de Spark *spark_session* para poder crear un spark dataframe con la información leída.

Para verificar la función no se usan pruebas unitarias, ya que se tiene que validar a nivel de base de datos. Es por eso que, para su evaluación, se hace consultas directamente a la base de datos por medio de SQL.

3.2.5. read_csv.py

Este archivo contiene una única función llamada *read_csv(csv_file_name,spark_session)* la cual hace una carga de archivos CSV en Spark dataframes. Para esto, se le pasan tanto el nombre del archivo como una sesión de Spark.

Esta función es critica para la ejecución general del programa, por lo tanto, la ejecución en si del programa es la validación de la función, y es por eso que no cuenta con funciones unitarias.

3.3. Folder unitary_tests

Contiene diferentes archivos Python con todas las funciones unitarias de pruebas para algunas de las funciones mencionadas anteriormente:

3.3.1. test_data_integrity.py

Contiene las siguientes pruebas unitarias para las funciones dentro de data_integrity.py:

1. **test_count_nan(spark_session)**: Prueba un conteo correcto de None dentro de un dataframe de pruebas.
2. **test_fill_nan_with_integer(spark_session)**: Prueba para sustituir None por un integer en una columna específica.
3. **test_fill_nan_with_string(spark_session)**: Similar a la prueba anterior, pero esta vez cambiando el None por un string.
4. **test_fill_nan_with_mean(spark_session)**: Prueba la función de sustitución de None por el valor de la media de una columna.
5. **test_fill_nan_non_numerical(spark_session)**: Prueba el modo de fallo de la función cuando se pasa como atributo una columna que no es algún tipo numérico.
6. **test_correct_date(spark_session)**: Prueba para la transformación de diferentes formatos de fecha por el deseado.

3.3.2. test_data_transformation.py

Contiene las siguientes pruebas unitarias para las funciones dentro de data_transformation.py:

1. **test_lower_case_string(spark_session)**: Prueba para la trasformación a minúsculas.
2. **test_dataframe_joiner_title(spark_session)**: Prueba de la unión de dos dataframes de prueba por medio del título.
3. **test_dataframe_joiner_title_missing_column(spark_session)**: Prueba el modo de fallo para cuando una de las columnas importantes no está presente.

3.4. build_run_docker.bat

Este archivo .bat es para el uso de este programa en Windows, y lo que hace es ejecutar secuencialmente diferentes comandos de consola. Los comandos usados en este archivo serán explicados en la sección de *Ejecución de la solución*.

3.5. commands_list.txt

Es un archivo de texto plano que contiene todas las lineas necesarias para crear el ambiente necesario en Docker, correr pruebas unitarias, ejecutar el ETL, levantar Jupyter y accesar las bases de datos en POSTGRES. Todos los comandos se explican en la sección de *Ejecución de la solución*.

3.6. Dockerfile

Es la imagen de Docker utilizada para esta solución, y es la imagen provista por el profesor llamada Dockerfile_full.

3.7. postgresql-42.7.3.jar

Contiene todas las librerías y funciones para interactuar con POSTGRES. El archivo cambio por el más actual (versión 47.7.3).

3.8. process_etl_YoksanVarela.py

Archivo de Python que se encarga de la extracción, transformación y carga de los datos para el análisis. En resumen, realiza los siguientes pasos:

1. Carga de los CSV en Spark Dataframes.
2. Limpieza de NaN y Null, ya sea con sustituciones o haciendo drop dependiendo de los casos.
3. Transformando todos los string a minúsculas.
4. Primer almacenaje de las dos tablas en la base de datos POSTGRES (y una carga subsecuente para verificación, esto podría eliminarse).
5. Se hace el joint de los dos dataframes usando el título del video juego.
6. Selección de las columnas a mantener y segundo almacenamiento en la base de datos (una sola tabla con todo junto; y también tiene un proceso de carga de regreso para verificación que podría ser omitido de igual forma).

3.9. solucion_proyecto_YoksanVarela.ipynb

Es el Jupyter Notebook donde se desarrolla toda la parte de análisis preliminar de los datos, entrenamiento y prueba de dos modelos de aprendizaje automático, que en este caso son Logistic Regression y Random Forest, y finaliza con una comparación de los modelos y conclusiones. El cuaderno en si se explica por si solo, por lo tanto, no se va a explicar en este documento, **pero es importante recalcar que en este Notebook esta el análisis de los resultados.**

4. Ejecución de la solución

Para poder ejecutar toda la solución, es necesario correr los siguientes comandos en el orden definido a continuación:

4.1. Creación de la imagen base de Docker

```
docker build --tag proyecto_yoksanvarela .
```

4.2. Crear una red interna en el Docker

Esto es para poder hacer que tanto el contenedor base con el contenedor de POSTGRES se puedan hablar entre sí y permitir las operaciones de lectura y escritura de las bases de datos por el Python para ETL como el notebook.

```
docker network create my_network
```

4.3. Ejecutar el contenedor de POSTGRES (en una sola línea)

```
docker run --name tarea3_db -e
POSTGRES_PASSWORD=testPassword -p 5433:5432
--network my_network -d postgres
```

4.4. Ejecutar el contenedor base (en una sola línea)

```
docker run --network my_network -p 8888:8888
-i -t proyecto_yoksanvarela /bin/bash
```

4.5. Correr las pruebas unitarias para data integrity (en el contenedor base)

```
pytest unitary_tests/test_data_integrity.py
```

4.6. Correr las pruebas unitarias para data transformation (en el contenedor base)

```
pytest unitary_tests/test_data_transformation.py
```

4.7. Ejecutar el programa MAIN que, en este caso, es el proceso de ETL (en el contenedor base, en una sola línea)

Es necesario incluir *-jars postgresql-42.7.3.jar* para que el driver de lectura y escritura de POSTGRES pueda ser utilizados por las funciones explicadas anteriormente.

```
spark-submit --jars postgresql-42.7.3.jar
process_etl_YoksanVarela.py
./datasets/all_video_games_info.csv ./datasets/vgchartz_sales_2024.csv
```

4.8. Levantar el proceso de Jupyter para ejecutar el Notebook de la solución (en el contenedor base)

```
jupyter notebook --ip=0.0.0.0 --port=8888 --allow-root
```

4.9. Acceder las bases de datos de POSTGRES (en el contenedor de POSTGRES)

```
psql -h host.docker.internal -p 5433 -U postgres
```

5. Bases de datos creadas

Al finalizar toda la ejecución de esta solución (incluyendo una ejecución completa del Jupyter Notebook), se van a haber creado cuatro bases de datos distintas:

5.1. video_game_info

Es la base da datos que contiene la información básica por video juego. El esquema es el siguiente:

1. Title: string
2. Release Date: date
3. Publisher: string
4. Genres: string
5. Product Rating: string
6. User Score: float

5.2. video_game_sales

Base de datos que contiene la informacion de las venta de los video juegos, asi como mas informacion relevante para el estudio. El esquema es el siguiente:

1. game_title: string
2. console: string
3. genre: string
4. game_publisher: string
5. total_sales: float

5.3. video_games_etl_completed

Es la base de datos que contiene la informacion de las otras dos bases de datos unificadas, y es la base datos a usar de entrada en el cuaderno de Jupyter. El esquema de esta base de datos es la siguiente:

1. game_title: string
2. console: string
3. publisher: string
4. release_date: date
5. genre: string
6. rating: string
7. user_score: float
8. total_sales: float

5.4. video_games_ml_ready

Es la base de datos final luego del análisis de los atributos. Es la información usada para entrenar y probar los dos modelos propuestos. El esquema es el siguiente:

1. console: string
2. publisher: string
3. genre: string
4. rating: string
5. label: string