Clusterizacion en línea utilizando River y Apache Kafka

Kräupl Morgenstern, Alain Wenzl[1] y Leon Marcos[1]

[1] Universidad Tecnológica Nacional FRC, Maestro M. Lopez esq. Cruz Roja Argentina, Ciudad de Córdoba Argentina

Abstract. If we train a machine learning model only once and never train it again, we will miss the information that this new data could provide us to improve the model. This is especially important in environments where behaviors change rapidly, such as online shopping. In this document we will analyze how to handle data flows in Python using Apache kafka and also how to train a clustering model (RFM k-means) using online learning techniques to adapt to incoming data flows using the River library.

Keywords: apache kafka, phyton, river, streaming data, stream learning, online data mining, online learning, incremental learning, clustering, dash.

1 Introducción

La generación de datos ha experimentado un crecimiento exponencial en la última década junto con el crecimiento de la infraestructura para manejarla. Cada aplicación de nuestro teléfono inteligente, los dispositivos *IoT* (Internet de las cosas), las interacciones de los usuarios con las aplicaciones (entre otras cosas) genera "toneladas" de datos. Se estima que en el año 2020 cada persona produjo 1,7 MB de datos por Segundo.

El enfoque convencional para la generación de modelos de *machine learning*, consiste en procesar los datos en lotes o fragmentos asumiendo que todos los datos están disponibles a la vez. Cuando un nuevo lote de datos está disponible, dichos modelos deben ser reentrenados desde cero. Lo que desea es un modelo que pueda predecir y aprender de nuevos ejemplos en algo cercano al tiempo real.

En múltiples aplicaciones los datos se generan continuamente, entonces podemos aplicar un enfoque diferente tratando los datos como un flujo. En otras palabras, como una secuencia infinita de elementos; los datos no se almacenan y los modelos aprenden continuamente una muestra de datos a la vez. Contar con este tipo de aprendizaje, llamado aprendizaje en línea, es especialmente importante en entornos donde los comportamientos cambian rápidamente como por ejemplo las compras que realizan los clientes a través de un *ecommerce*.

El proyecto se basa en la implementación de un algoritmo de minería de datos en línea obtenidos de un flujo transmitido por la plataforma Kafka. El resultado obtenido será visualizado en un dashboard interactivo para su posterior análisis.

2 Origen de los datos

Para la realización de este documento utilizamos un conjunto de datos (también llamado *dataset*) públicos de la empresa de comercio electrónico Olist Store. Esta es la tienda por departamentos más grande de los mercados brasileños. Conecta pequeñas empresas de todo Brasil con canales sin problemas y con un solo contrato. Esos comerciantes pueden vender sus productos a través de Olist Store y enviarlos directamente a los clientes mediante los socios logísticos.

Después de que un cliente compra el producto en Olist Store, un vendedor recibe una notificación para cumplir con ese pedido. Una vez que el cliente recibe el producto, o vence la fecha estimada de entrega, el cliente recibe una encuesta de satisfacción por correo electrónico donde puede dar una nota por la experiencia de compra y anotar algunos comentarios.

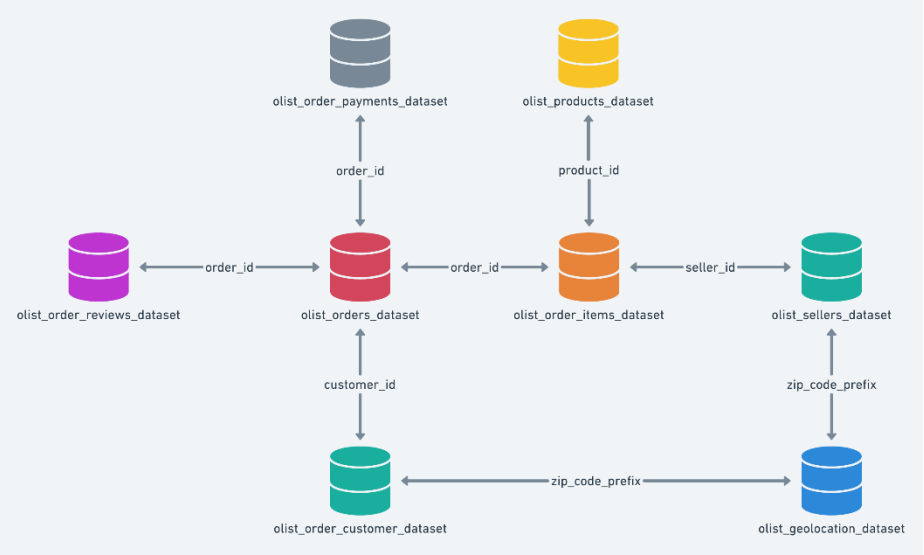

Fig. 1. Diagrama de entidad de relación provisto por Olist Store del *dataset*.

El *dataset* provisto está compuesto por ocho archivos csv que representan cada una de las tablas de la ***Fig. 1***, donde:

- olist_customers_dataset.csv: tiene información sobre el cliente y su ubicación.
- olist_geolocation_dataset.csv: tiene información de los códigos postales brasileños y sus coordenadas lat/lng.
- olist_order_items_dataset.csv: detalle de los artículos comprados dentro de cada pedido.
- olist_order_payments_dataset.csv: detalle de como pagaron los clientes el pedido.

- olist_order_reviews_dataset.csv: comentarios de los clientes luego de recibir el pedido.
- olist_orders_dataset.csv: detalle de todos los pedidos realizados.
- olist_products_dataset.csv: detalle de todos los productos vendidos por Olist.
- olist_sellers_dataset.csv: tiene información sobre los vendedores y su ubicación.
- product_category_name_translation.csv: tiene la información sobre las categorías de los productos vendidos.

El conjunto de datos tiene información de 100k pedidos de 2016 a 2018 realizados en múltiples mercados en Brasil. Sus características permiten ver un pedido desde múltiples dimensiones: desde el estado del pedido, el precio, el pago, el desempeño del flete, la ubicación del cliente y del vendedor, los atributos del producto y finalmente las reseñas escritas por los clientes. También publicamos un conjunto de datos de geolocalización que relaciona los códigos postales brasileños con las coordenadas lat/lng.

Estos son datos comerciales reales, se han anonimizado y las referencias a las empresas y socios en el texto de revisión se han reemplazado con los nombres de las grandes casas de *Game of Throne*.

2.1 Preprocesamiento

Antes de explorar nuestro conjunto de datos vamos a realizar unas transformaciones simples como ser:

1. Como primera medida generamos un único *data frame* a partir de los archivos csv provisto por Olist Store utilizando la librería para *Phyton* pandas.
2. Una vez que tenemos nuestro conjunto de datos en un único objeto *dataframe* de pandas, utilizamos el método *describe* para ver algunos detalles estadísticos básicos como percentil, media, estándar, etc.…
3. Convertimos cada una de las variables a los tipos de datos que corresponden. Como por ejemplo un variable con una fecha cuyo tipo sea *string* o *object* la convertimos en *datetime.* Procedemos de forma similar para los tipos de variable numérica.
4. Sustituimos los valores nulos o vacíos por la media de la variable en cuestión.
5. Eliminamos todas las filas duplicadas llamando al método *duplicated* de nuestro objeto *dataframe.*
6. Creamos nuevas características o variables a partir de las ya existentes, como ser:
    - order_process_time para ver cuánto tiempo llevará iniciar el pedido hasta que los artículos son aceptados por los clientes
    - order_delivery_time para ver cuánto tiempo se requiere de envío para cada pedido
    - order_time_accuracy para ver la diferencia entre el tiempo estimado de envió y el tiempo que realmente demoro.

- order_approved_time para ver cuánto tiempo tomará desde el pedido hasta la aprobación
- review_send_time para averiguar cuánto tiempo se envió la encuesta de satisfacción después de recibir el artículo.
- review_answer_time para ver la demora en hacer un comentario por parte del cliente desde que le llego el pedido
- product_volume para ver el volumen de cada paquete enviado

2.2 Un poco de EDA (Exploratory Data Analysis)

Con el objetivo de conocer y comprender más a fondo el conjunto de datos realizamos un análisis exploratorio sobre el mismo intentando responder algunas preguntas. Para realizar los gráficos se utilizó la librería seaborn [2].

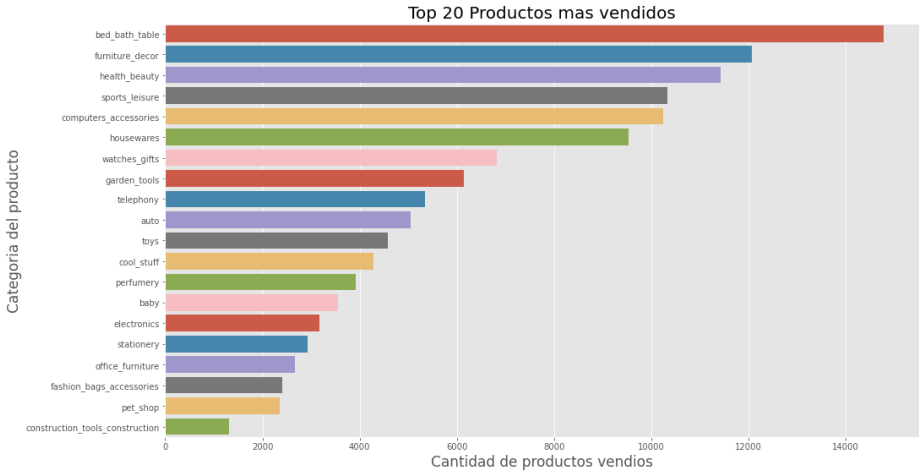

Fig. 2. ¿Qué productos tienen más demanda?

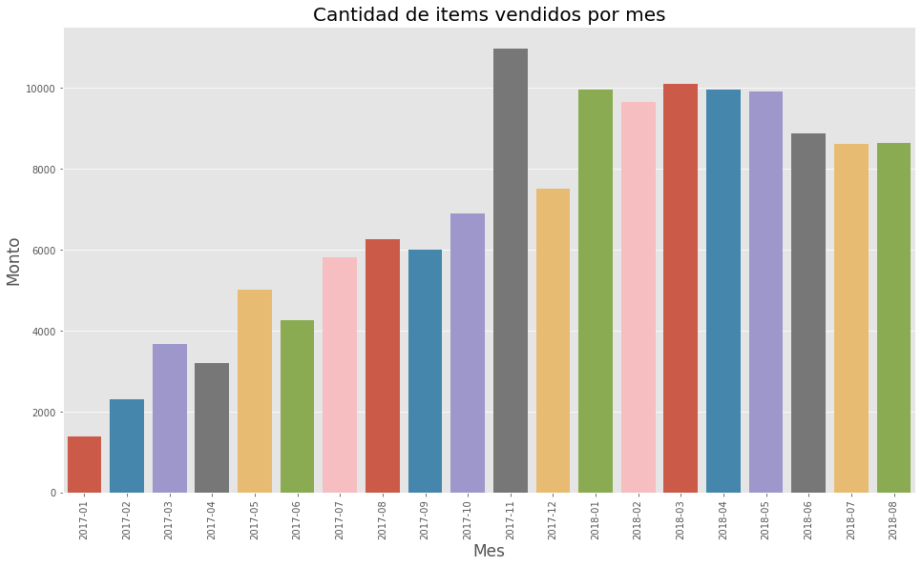

Fig. 3. ¿Cuántos ítems fueron vendidos por mes?

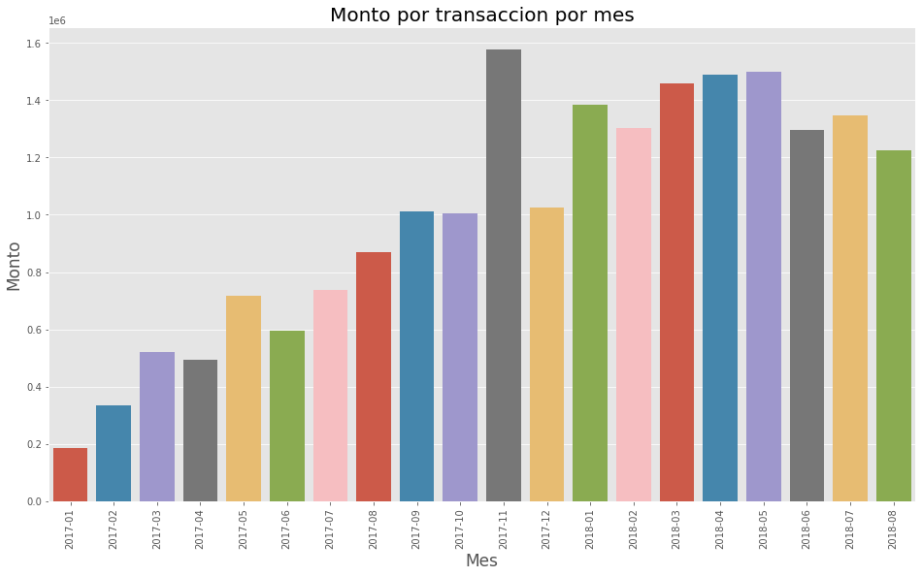

Fig. 4. ¿Cuál fue el monto por transacción por mes?

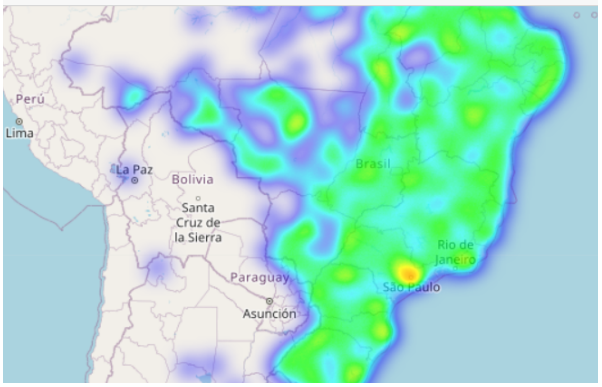

Fig. 5. ¿Dónde se encuentran los clientes?

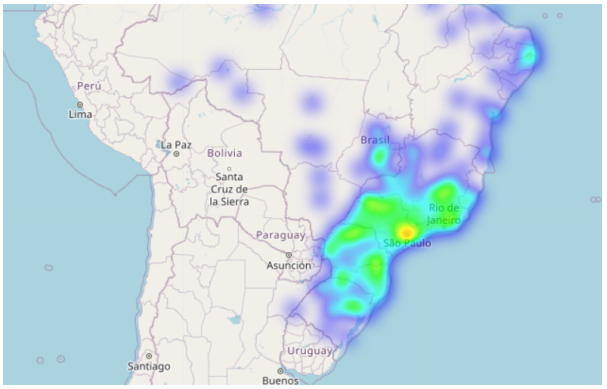

Fig. 6. ¿Dónde se encuentran los vendedores?

3 Streaming de datos

Kafka es una plataforma de streaming de datos utilizada para construir aplicaciones que requieran centralizar la transmisión de grandes volúmenes de datos entre sus componentes. Permite codificarla de tal manera que su despliegue sea distribuido y tolerante a fallos.

Su funcionamiento se basa en el patrón arquitectónico Publish & Suscribe, sobre el cual diferentes productores y consumidores envían y reciben mensajes.

Un stream de mensajes esta definido por un tópico. Un productor puede publicar mensajes a un tópico. Los mensajes publicados luego son guardados en un conjunto de servidores llamados brokers. Un consumidor se puede suscribir a uno o más tóicos de bokers para luego consumir los mensajes suscriptos [2].

Zookeeper es un servicio centralizado e imprescindible para el óptimo funcionamiento de Kafka, al cual envía notificaciones en caso de cambios, como lo son al momento de la creación, borrado de un nuevo tópico, caída o levantameinto de un broker, etc.

Su labor principal es gestionar los brokers de Kafka, manteniendo un listado con sus respectivos metadatos para facilitar mecanismos de health checking.

4 Aprendizaje en línea o incremental

Los métodos de aprendizaje automático ofrecen tecnologías particularmente poderosas para inferir información estructural a partir de datos dados. Aun así, la mayoría de las aplicaciones actuales se restringen a la configuración clásica por lotes: los datos se proporcionan antes del entrenamiento, por lo tanto, la selección del modelo, la optimización de meta parámetros y el entrenamiento se basan en un conjunto datos estáticos. El aprendizaje incremental, por el contrario, se refiere a la situación de adaptación continua del modelo basad en datos que llegan constantemente (*data stream*). Esta configuración está presente siempre que los sistemas actúen de forma autónoma como en robótica o conducción autónoma. Además, el aprendizaje en línea se vuelve necesario en escenarios interactivos donde los ejemplos de capacitación se proporcionada en base a la retroalimentación humana a lo largo del tiempo. Finalmente, algunos conjuntos de datos o *dataset*, aunque sean estáticos, pueden llegar a ser tan grandes que de hecho se tratan como un flujo de datos.

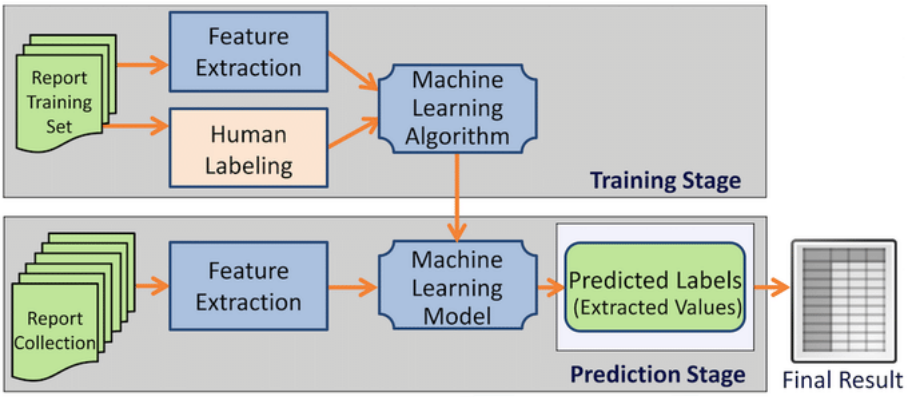

Fig. 7. Aprendizaje por lotes (Ofline)

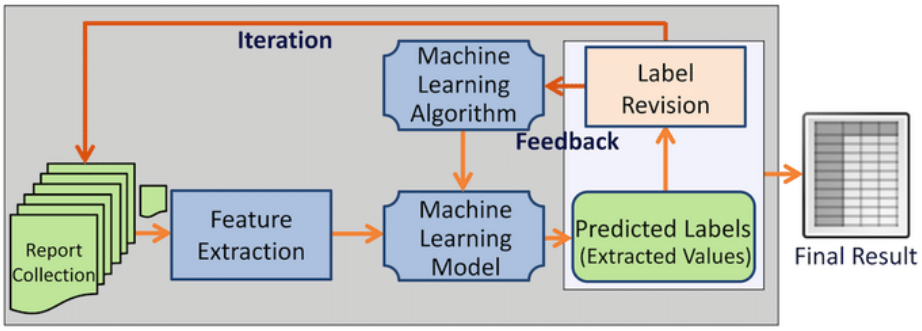

Fig. 8. Aprendizaje incremental (Online)

4.1 River

River [4] es una libreria para Phyton para el aprendizaje automático en línea, también conocido como aprendizaje incremental. Es el resultado de una fusión entre Creme [5] y Scikit-multiflow [6]. El aprendizaje en línea es un régimen de aprendizaje automático en el que un modelo aprende una observación a la vez.

A diferencia del aprendizaje por lotes, donde todos los datos se procesan a la vez. El aprendizaje incremental es deseable cuando los datos son demasiado grandes para caber en la memoria, o simplemente cuando desea manejar datos de transmisión. Además de muchos algoritmos de aprendizaje automático en línea, River proporciona utilidades para extraer características de un flujo de datos.

Algunas de las ventajas de esta librería son:

- **Incremental**: Todas las herramientas de la biblioteca se pueden actualizar con una sola observación a la vez y, por lo tanto, se pueden utilizar para procesar datos de transmisión.
- **Adaptada**: En el entorno de transmisión, los datos pueden evolucionar. Los métodos adaptativos están diseñados específicamente para ser robustos frente a la deriva de conceptos en entornos dinámicos.
- **De propósito general**: River atiende diferentes problemas de *machine learning*, incluida la regresión, la clasificación y el aprendizaje no supervisado. También se puede utilizar para tareas ad hoc, como la computación de métricas en línea y la detección de desviaciones de conceptos.
- **Eficiente y fácil de usar**: Las técnicas de transmisión manejan de manera eficiente recursos como la memoria y el tiempo de procesamiento dada la naturaleza ilimitada de los flujos de datos. River está diseñado para usuarios con cualquier nivel de experiencia.

Todos los modelos predictivos realizan dos funciones básicas: aprender y predecir. Entrenaremos el modelo creado llamando al método learn_one, el cual actualiza el estado interno del modelo. Por otro lado, los modelos proporcionan predicciones a través del método predict_one (para algoritmos de clasificación, regresión, y

agrupación), predict_proba_one (para algunos clasificación) y score_one (detección de anomalías).

5 Clusterización

El *clustering* es una tarea que tiene como finalidad principal lograr el agrupamiento de conjuntos de objetos no etiquetados, para lograr construir subconjuntos de datos conocidos como *Clusters*. Cada *cluster* está formado por una colección de objetos o datos que a términos de análisis resultan similares entre si, pero que poseen elementos diferenciales con respecto a otros objetos pertenecientes al conjunto de datos y que pueden conformar un *cluster* independiente.

Este tipo de proceso es aplicado en modelos de *machine learning* de tipo no supervisado. Gracias a su implementación el sistema puede analizar los datos, realizar la tarea y encontrar los posibles errores dentro de su funcionamiento. El *Clustering*, en este caso sirve para segmentar datos en grupos de dimensiones similares en base a características para facilitar este proceso.

5.1 K-means con análisis RFM

Realizaremos la clusterizacion de los clientes utilizando *k-means* [7] con análisis RFM [8] (*Recency, Frequency, Monetary value*) que es uno de los métodos de segmentación de clientes más sencillos de implementar, y al mismo tiempo uno de los que mejores resultados aportan a corto plazo. Para ello crearemos tres nuevas características o variables a nuestro conjunto de datos:

- *recency*: ¿Cuándo fue la última vez que un Cliente me compró algo?
- *frecuency*: ¿Cuántas veces me ha comprado un Cliente en el periodo de análisis?
- *monetary value*: ¿Cuál ha sido el valor monetario agregado de un Cliente en dicho periodo?

Los algoritmos de aprendizaje automático son sensibles al rango y distribución de los valores de las variables. Los datos atípicos pueden estropear y engañar el proceso de entrenamiento, lo que da como resultado tiempos de entrenamiento más largos, modelos menos precisos y, en última instancia, resultados más pobres. utilizaremos el método IQR [9] para identificar estos valores atípicos (*outliers*) y eliminarlos.

La última transformación que aplicaremos a nuestros datos es una normalización [10]. El objetivo de la normalización es cambiar los valores de las columnas numéricas en el conjunto de datos a una escala común, sin distorsionar las diferencias en los rangos de valores.

Utilizaremos con el método del codo [11] (*elbow curve*) el número k óptimo de clusters para entrenar nuestro modelo de *k-means*. El método del codo traza el valor de la función de costo producida por diferentes valores de k. Si k aumenta, la distorsión promedio disminuirá, cada grupo tendrá menos instancias constituyentes y las instancias estarán más cerca de sus respectivos centroides. Sin embargo, las mejoras en la distorsión promedio disminuirán a medida que aumente. El valor de k en el que la mejora en la distorsión disminuye más se llama codo.

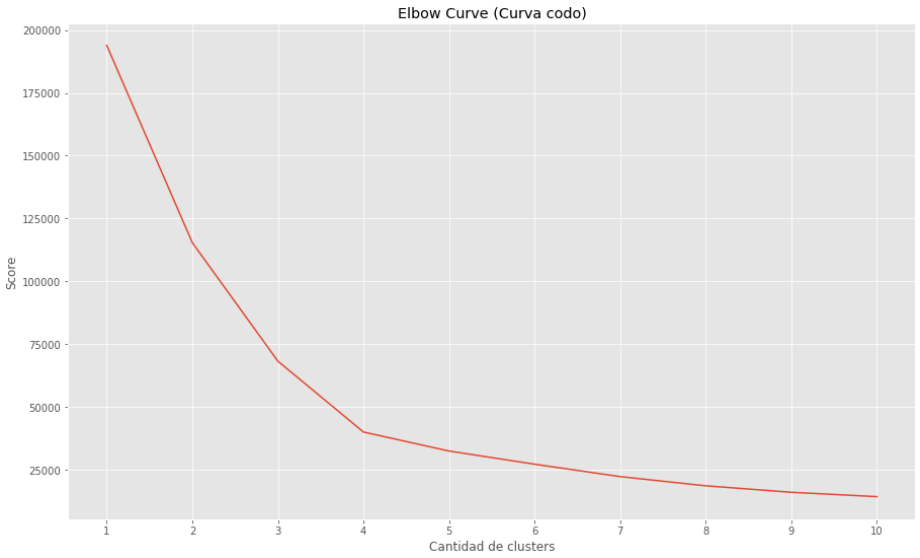

Fig. 9. *Elbow curve*. Vemos que los posibles valores de K son 4 o 5

5.2 Validación del modelo

Las dos categorías más importantes para la validación de *clustering* son la validación externa y la validación interna. La principal diferencia es si se usa o no información externa para la validación, es decir, información que no es producto de la técnica de agrupación utilizada.

A diferencia de técnicas de validación externas, las de validación interna miden el *clustering* únicamente basadas en información de los datos. Evalúan que tan buena es la estructura del *clustering* sin necesidad de información ajena al propio algoritmo y su resultado. Como la validación externa mide la calidad del agrupamiento conociendo información externa de antemano, es principalmente usada para escoger un algoritmo de *clustering* óptimo sobre un *dataset* específico. Las métricas de validación interna pueden usarse para escoger el mejor algoritmo de *clustering*, así como el número de clúster óptimo sin ningún tipo de información adicional. En la práctica, la información externa, como los labels de las clases , por lo general no se encuentra disponible en muchos escenarios de aplicación.

Validación Interna. Como el objetivo del *clustering* es agrupar objetos similares en el mismo clúster y objetos diferentes ubicarlos en diferente clúster, las métricas de validación interna están basadas usualmente en los dos siguientes criterios:

- Cohesión: El miembro de cada clúster debe ser lo más cercano posible a los otros miembros del mismo clúster.

- Separación: Los clústeres deben estar ampliamente separados entre ellos. Existen varios enfoques para medir esta distancia entre clúster: distancia entre el miembro más cercano, distancia entre los miembros más distantes o la distancia entre los centroides.

Validaremos la evolución de nuestro modelo utilizando las siguientes métricas:

- Cohesion: Distancia media desde los puntos a sus centroides de grupo asignados. Mientras más chico mejor.
- Separation: Distancia media de un punto a los puntos asignados a otros grupos. Mientras mas grande mejor.
- SSW (*Sum-of-Squares Within Clusters*): medida interna especialmente usada para evaluar la Cohesión de los clústeres que el algoritmo de agrupamiento generó. Mientras más grande mejor.
- SSB (*Sum-of-Squares Between Clusters*) : es la media ponderada de los cuadrados de las distancias entre los centros de los *clusters* al valor medio de todo el conjunto de datos. Mientras más grande mejor.
- BIC (*Bayesian Information Criterion*): es un criterio para la selección de modelos entre un conjunto finito de modelos. Cuando el ajuste de modelos, es posible aumentar la probabilidad mediante la adición de parámetros, pero si lo hace puede resultar en sobreajuste. Tanto el BIC y AIC resuelven este problema mediante la introducción de un término de penalización para el número de parámetros en el modelo, el término de penalización es mayor en el BIC que en el AIC. Mientras más grande mejor.
- Silhouette: es la relación entre la cohesión y las distancias promedio desde los puntos hasta su segundo centroide más cercano. Recompensa el algoritmo de agrupamiento donde los puntos están muy cerca de sus centroides asignados y lejos de cualquier otro centroide, es decir, resultados de agrupamiento con buena cohesión y buena separación. Mientras más chico mejor.
- XieBeni: define la separación entre grupos como la distancia mínima al cuadrado entre los centros de los grupos, y la compacidad intra-grupo como la distancia cuadrática media entre cada objeto de datos y sus centros de clúster. Mientras más chico mejor.

Nota: para consultar el detalle del EDA, ETL y la implementación del modelo k-means RFM ver notebook [12].

6 Visualización

Dash es un framework de Python para construir aplicaciones web dedicadas al análisis de datos.

Escrita en una capa superior a Flask, Plotly y React, Dash es ideal para la construción de aplicaicones de visualización de datos con interfaces de usuario personalizadas.

Las aplicaciones desarrolladas se renderizan en el navedaor, esto conlleva que podemos tener nuestra aplicación o servicio desplegado en un servidor, y posteriormente podemos compartir, mediante URL, con varias personas nuestro dashboard o cuadro de mando.

7 Arquitectura final

La arquitectura de solución propuesta e implementada tiene como actores principales a componentes nombrados anteriormente, como lo son Kafka para soportar la transmisión de datos desde sus origenes y la implementación del modelo k-means mediante la librería River.

- Fuente de datos: Conjunto de csv exportados de un modelo de datos transaccional que se encarga de soportar el negocio.
- Producer: Implementación de un productor de mensajes a transmitir en Kafka. Desarrollado utilizando el lenguaje de programación Python.
- Kafka: Plataforma de transmisión de mensajes encargada de conectar productores y consumidores de datos mediante brokers y tópicos.
- Zookeeper: Herramienta utilizada para gestionar el sistema distribuido de Kafka.
- Consumer: Implementación de un consumidor de mensajes a recibir de Kafka. Desarrollado utilizando el lenguaje de programación Python.
- Mining-Server: Implementación de una API Rest encargada de recibir los datos a procesar con el algoritmo de minería de datos. Desarrollado utilizando el framework Flask.
- Mining-ETL: Componente encargado del depurado de los datos provenientes del stream y recibidos por el Server. Transforma los datos y genera variables a clusterizar.
- Mining-Clustering: Implementación del algoritmo k-means RFM utilizando la librería River.
- Visualización: Componente encargado de visualizar resultados obtenidos por el agrupamiento. Implementado utilizando el framework Dash, el cual se encarga de ofrecer un dashboard interactivo para visualizar el comportamiento de los centroides y métricas analizadas.

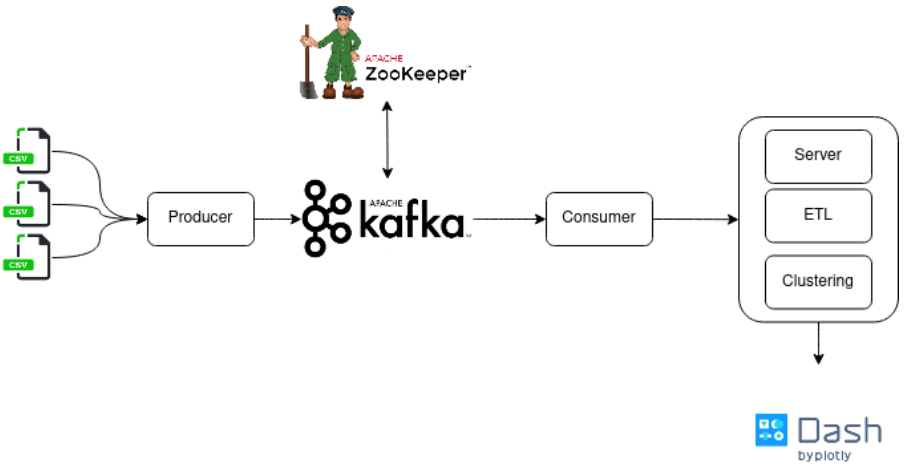

Fig. 10. Arquitectura propuesta.

8 Reproducibilidad

Requisitos

- Docker 20.10.2 o superior [12]
- Docker-compose 1.25.0 o superior [13]
- Git 2.8.0 o superior [14]

Pasos:

1. Clonar respositorio de código fuente [15] utilizando Git:
   - git clone https://github.com/alaain04/maestria-sistemas-dm.git
2. Moverser al directorio raiz del repositorio descargado:
   - cd maestria-sistemas-dm
3. Ejecutar docker compose para iniciar el proceso:
   - docker-compose up –build
4. Ingresar al navegador web la direccion http://localhost:8050, donde se encontrará desplegado el dashboard.

Referencias

1. Jacob Montiel: River: machine learning for streaming data in Python, https://arxiv.org/pdf/2012.04740v1.pdf
2. Jay Kreps, Neha Narkhede, Jun Rao: Kafka: a Distributed Messaging System for Log Processinghttps://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf
3. Seaborn, https://seaborn.pydata.org, último acceso 2021/05/12.
4. River, https://riverml.xyz/latest/, último acceso 2021/05/12.
5. Creme, https://pypi.org/project/creme/, último acceso 2021/05/12.
6. Scikit-multiflow, https://scikit-multiflow.github.io/, ultimo acceso 2021/05/12.
7. K-means, https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1, último acceso 2021/05/12.
8. Mohammadreza Tavakoli: Customer Segmentation and Strategy Development based on User Behavior Analysis, RFM model and Data Mining Techniques: A Case Study, https://www.researchgate.net/publication/330027350_Customer_Segmentation_and_Strategy_Development_Based_on_User_Behavior_Analysis_RFM_Model_and_Data_Mining_Techniques_A_Case_Study
9. IRQ, https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097, último acceso 2021/05/12.
10. Normalization, https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0, último acceso 2021/05/12.
11. Elbow Curve, https://www.scikit-yb.org/en/latest/api/cluster/elbow.html, último acceso 2021/05/12.
12. Notebook, https://drive.google.com/drive/folders/1_wMWrZQk02mFcLgYv5OJGgOt1morgW7U?usp=sharing, último acceso 2021/05/12.
13. Docker compose install, https://docs.docker.com/compose/install/, último acceso 2021/05/14.
14. Docker install, https://docs.docker.com/engine/release-notes/, ultimo acceso 2021/05/14
15. Git install, https://git-scm.com/book/en/v2/Getting-Started-Installing-Git, último acceso 2021/05/14.
16. Codigo fuente Github, https://github.com/alaain04/maestria-sistemas-dm, último acceso 2021/05/14.