Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Análisis y Visualización de Datos Masivos

Sistema recomendador en tiempo real para ofertas comerciales en el sector bancario

Trabajo Fin de Máster

Tipo de trabajo: Desarrollo de software

Presentado por: Alvarado Dueñas, Gustavo Andrés

Director/a: Torralba Elipe, Guillermo

Resumen

En los últimos años ha habido un aumento en el uso de sistemas recomendadores fuera de las industrias tradicionales como el comercio electrónico o el entretenimiento. Tal es el caso del sector bancario donde existen numerosas oportunidades de aplicación como por ejemplo la recomendación de productos. Sin embargo, los principales obstáculos para su implementación son la carencia de ratings debido a que los clientes no suelen calificar un producto bancario y la imposibilidad de poder dar una recomendación a un cliente nuevo por falta de datos. Para solucionar el primer problema, el presente trabajo propone un algoritmo para calcular el rating implícito de un producto y con ello alimentar un modelo basado en filtrado colaborativo. Para el segundo caso, se realiza un modelo de clustering para asignar al cliente a un grupo de personas con similares características y con ello poder generar una recomendación. Ambos modelos son soportados por una arquitectura que hace posible la predicción en tiempo real para la generación de una oferta comercial en el momento oportuno. Para validar la propuesta se utilizaron datos reales de transacciones hechas con tarjeta correspondientes a más de 70 mil clientes de un reconocido banco peruano y se calculó el error en la predicción usando las métricas MAE y RMSE. Los resultados fueron bastante aceptables ya que ambos modelos superaron en más de 30% a un predictor aleatorio y se demostró que el enfoque propuesto para calcular el rating implícito considerando la frescura, frecuencia y monto de transacciones obtuvo mejores resultados que otro estudio que solo consideró una de esas variables.

Palabras Clave: sistemas recomendadores, filtrado colaborativo, clustering, arquitectura real-time, banca, rating implícito

Abstract

In recent years, the use of recommender systems outside of traditional industries such as e-commerce or entertainment has increased. This is the case of banking sector where there are many opportunities like product recommendation. However, there are two main problems to its implementation: data sparsity due to the lack of ratings because customers do not usually rate a banking product and cold start which is the impossibility to give a recommendation to a new customer. In order to solve the first problem, this work proposes an algorithm to calculate an implicit rating of a product to use it in a collaborative filtering model. On the other hand, a clustering model is carried out to assign the client to a group of people with similar characteristics and thereby generate a recommendation. Both models are supported by an architecture that enables real-time prediction for the generation of a right time commercial offer. To validate the proposal, real data from card transactions corresponding to more than 70,000 clients of a well-known Peruvian bank were used, and prediction error was calculated using MAE and RMSE metrics. The results were quite acceptable because both models outperformed a random predictor by more than 30% and it was demonstrated that the proposed approach to calculate implicit rating using recency, frequency and monetary variables got better results than other study that only considered one of those variables.

Keywords: recommender systems, collaborative filtering, clustering, real-time architecture, banking, implicit rating

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

La evolución de las tecnologías de la información ha cambiado la manera de hacer negocios y el sector bancario no es ajeno a ello. Los sistemas que inicialmente solo soportaban la carga de transacciones durante los horarios de oficina ahora se ven sobrepasados con las grandes cantidades de datos que generan los clientes desde diferentes fuentes de manera continua.

Estamos en una era donde los clientes, en especial los “nativos digitales”, demandan respuestas rápidas, por ello es muy importante que las empresas tengan la capacidad de capturar, analizar y extraer patrones de los datos para poder ofrecer el producto más adecuado a un cliente específico en el momento oportuno.

Sin embargo, son pocas las entidades que logran este objetivo ya que la mayoría aún utilizan estrategias de marketing tradicional centrado en el producto en lugar del cliente, lo cual deriva en campañas comerciales masivas muy poco personalizadas que trae como consecuencia bajas tasas de respuesta a las ofertas y clientes insatisfechos.

Ante esto surge el concepto denominado “marketing en el tiempo justo” (o “right time marketing”) que no es otra cosa que la evolución del marketing tradicional de interrupción basado en campañas masivas generadas en modo batch, a un marketing centrado en el cliente con campañas personalizadas a sus necesidades y en tiempo real (Goldstein & Lee, 2005). Esto, hasta hace algunos años, era solo una aspiración, pero con el surgimiento de las nuevas tecnologías ya es posible hacerlo realidad.

En este sentido, las tecnologías de big data permiten manejar los grandes volúmenes de datos que los clientes están generando en todo momento, acceder a nuevas fuentes que antes no eran explotadas, procesarlas y ponerlas a disposición de los modelos analíticos para realizar predicciones, todo esto en tiempo real. Los resultados de este proceso son la base para la toma de decisiones desde un punto de vista comercial.

Las empresas que adoptan esta nueva forma de marketing y las nuevas tecnologías que hacen posible su implementación tienen una ventaja competitiva importante que se traduce en oportunidades de negocio para atraer nuevos clientes, aumentar la satisfacción y evitar la fuga de los clientes actuales, además de obtener mayores beneficios derivados de estrategias de cross-selling y up-selling de los productos (Jason Lu, et al., 2022).

1.1 Motivación

Como parte de la transformación digital en el sector, los principales bancos ya vienen utilizando las nuevas tecnologías en sus procesos comerciales. Por ejemplo, ya es común que se utilicen algoritmos de machine learning para poder predecir la propensión de un cliente para comprar un producto específico; sin embargo, estos procesos aún dependen de datos que se cargan en batch lo que hace que muchas veces una oferta llegue al cliente cuando este ya no la necesita.

Una explicación para esto es que la mayoría de estos procesos fueron creados en los data warehouses corporativos que en su momento fueron los únicos repositorios que permitían almacenar un histórico de los datos generados por el negocio. En este escenario, era común que los datos fuesen actualizados al final del día y se tomaban decisiones reactivas con los datos disponibles (Megargel et al., 2018).

A esto se le suma la carencia de una arquitectura que esté orientada a eventos, que permita la captura de los datos generados por las interacciones de los clientes con el banco desde cualquier canal y que esté disponible para su consumo en tiempo real por las diferentes aplicaciones o procesos, entre ellos el de generación de campañas comerciales (Shankararaman & Megargel, 2013).

Para poder ser competitivo en la era digital ya no es suficiente analizar los datos históricos y determinar qué pasó, sino que es necesario dar el siguiente paso y poder predecir lo que pasará. En este sentido, determinar la siguiente mejor oferta (lo que se conoce como “Next Best Offer” o NBO) para un cliente específico en tiempo real se ha convertido en la principal estrategia de marketing de los bancos en la actualidad (Davenport et al., 2011).

1.2 Planteamiento del trabajo

Para poder implementar un modelo de NBO en una organización, Davenport et al. (2011) plantean una metodología de 4 pasos:

- Definir el objetivo: es decir lo que se quiere lograr, por ejemplo: incrementar ventas, atraer nuevos clientes, aumentar la retención de clientes etc.

- Recolectar los datos: capturar e integrar los datos relativos a los clientes (perfil, transacciones, actividad e interacciones), características de los productos e información contextual como la geolocalización.
- Analizar y ejecutar: identificar la tecnología correcta para analizar y predecir la mejor oferta para el cliente y el canal más adecuado para comunicarla.
- Aprender y evolucionar: se refiere a implementar mecanismos para monitorear el funcionamiento del modelo.

El presente trabajo se centrará en los puntos 2 y 3 de la metodología mencionada. Para la recolección de datos se añade la característica del tiempo real y para el análisis y ejecución se revisará la tecnología necesaria para la captura de los datos, procesamiento, así como el detalle del modelo de machine learning que realizará la recomendación.

El objetivo general de la propuesta es diseñar un sistema recomendador en tiempo real que permita a los equipos comerciales del banco gestionar una oferta personalizada para un cliente en base a sus hábitos de consumo con su tarjeta de crédito o débito y a otros atributos como: datos demográficos, ingresos, productos contratados, saldos de deuda, clasificación crediticia etc.

1.3 Estructura de la memoria

El presente TFM se encuentra organizado de la siguiente manera:

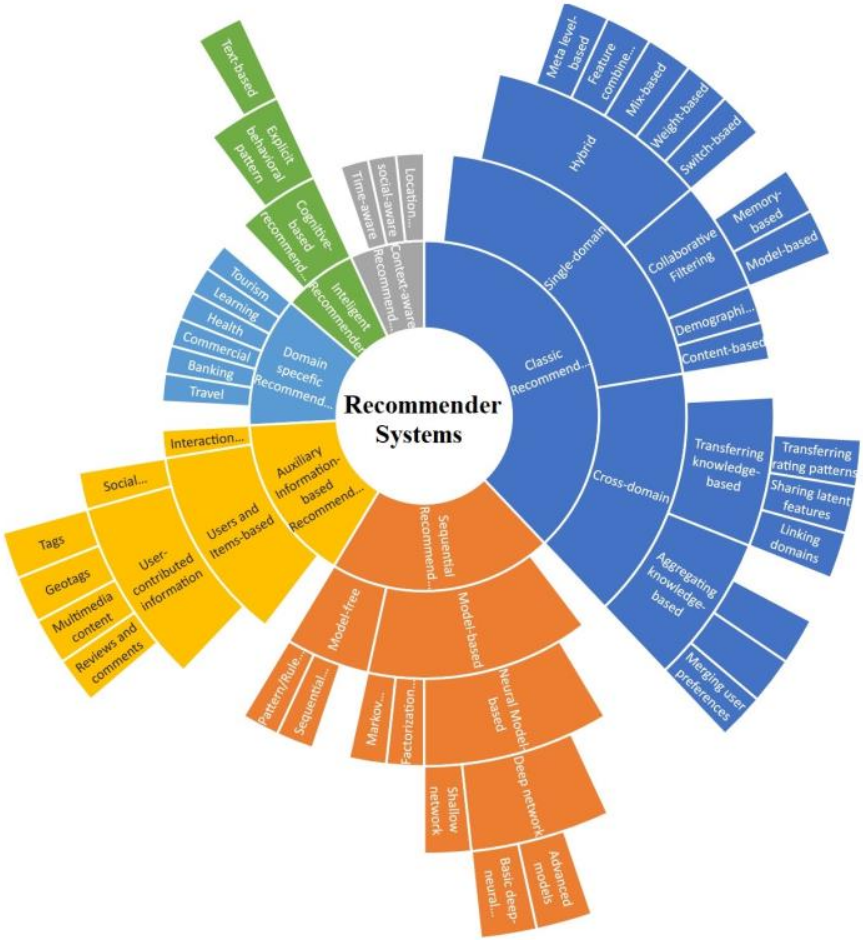
- En el capítulo 2, se da un contexto sobre el tema y se revisan trabajos relacionados sobre sistemas recomendadores, sus aplicaciones en el sector financiero y las tecnologías utilizadas para la captura y procesamiento de datos en tiempo real.
- Luego, en el capítulo 3, se repasan los objetivos que se pretenden cumplir y la metodología que se utiliza para cada etapa de la solución planteada.
- En los capítulos 4 y 5 se describe a detalle la contribución, la cual se divide en el desarrollo del sistema recomendador como tal y el planteamiento de la arquitectura para soluciones en tiempo real. Además, se presentan los resultados obtenidos para evaluar si realmente la propuesta cumple con los objetivos descritos en el capítulo 3.
- El capítulo 6 resume las conclusiones del trabajo y las líneas que quedan abiertas para futuras investigaciones.

2. Contexto y estado del arte

Un sistema recomendador es un programa que busca recomendar el producto o servicio más adecuado para un usuario tomando en cuenta la información más relevante obtenida de una fuente de datos privada, pública, redes sociales, IoT (Internet de las cosas) entre otras (Beheshti et al., 2020).

Existen numerosos estudios sobre sistemas recomendadores debido a su aplicabilidad en prácticamente todas las industrias y como se puede ver en la Figura 1 su clasificación es bastante amplia.

Figura 1. Clasificación de sistemas recomendadores



Fuente: (Beheshti et al., 2020)

El presente capítulo se centra en los sistemas recomendadores clásicos orientados a un dominio específico (en este caso el sector bancario) por lo cual se detallan los conceptos de filtrado colaborativo, filtrado basado en contenido y sistemas híbridos destacando las características, ventajas y desventajas de cada uno.

Luego se comentan algunos trabajos relacionados que muestran la utilidad de los sistemas recomendadores fuera de las industrias tradicionales como el comercio electrónico, contenido musical, contenido de video o redes sociales. Tal es el caso del sector financiero donde se mencionan aplicaciones en los dominios de préstamos, seguros, bienes inmuebles, valor de acciones, gestión de portafolio de inversiones y banca. Se destacan las aplicaciones en este último dominio ya que corresponden al ámbito en el cual se desarrolla este trabajo.

Asimismo, una de las partes más importantes de la propuesta es la posibilidad de dar recomendaciones en tiempo real, por ello se mencionan algunos estudios sobre sistemas recomendadores con esta característica poniendo énfasis en las arquitecturas y tecnologías utilizadas para este fin.

2.1 Tipos de sistemas recomendadores

En su estudio, Sharaf et al. (2022) clasifican a los sistemas recomendadores de la siguiente manera:

2.1.1 Filtrado colaborativo

Se basa en las preferencias de los usuarios y sus opiniones sobre diversos ítems. Este método asume que, si dos personas tienen la misma opinión sobre un ítem, tendrán la misma opinión sobre otros. Es muy utilizado y puede dar predicciones acertadas sin necesidad de conocer el ítem, pero sufre de 3 problemas principales:

- “Cold start”: se traduce en problemas para recomendar ítems a un usuario nuevo o recomendar ítems nuevos.
- Escalabilidad: la latencia de respuesta, que en muchos casos se espera que sea en tiempo real, puede verse disminuida con el incremento en la cantidad de datos que el sistema debe procesar.
- Escasez de ratings (“data sparsity”): en grandes industrias como la del comercio electrónico, la cantidad de usuarios que califican los productos es relativamente baja con respecto al total.

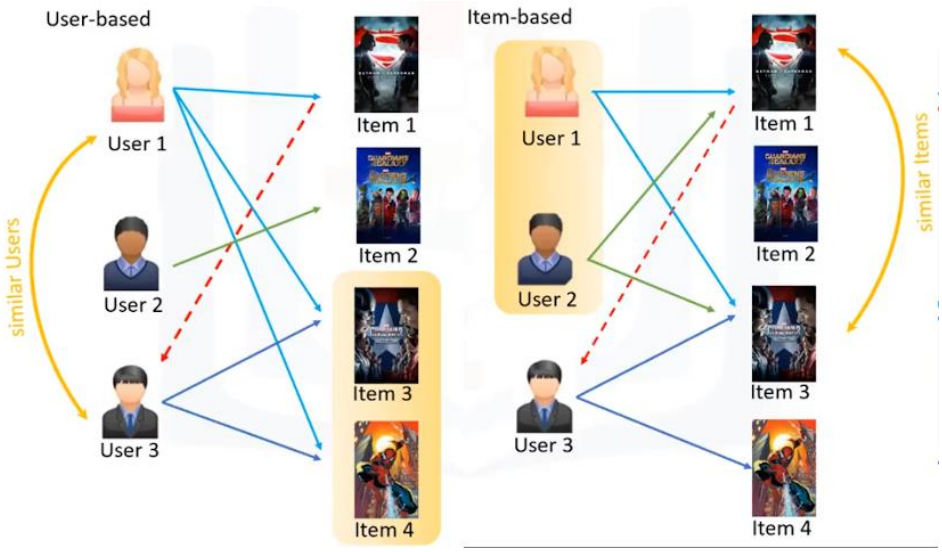
Dentro del filtrado colaborativo existen 2 categorías que son:

- Basado en memoria: se realiza de acuerdo a las preferencias de los usuarios sobre los ítems expresada a través de una calificación (rating) y basa la recomendación de acuerdo a la similitud entre los usuarios o los ítems. Por ejemplo, en la Figura 2 se

muestra que los usuarios 1 y 3 son similares, por lo tanto, si el usuario 1 valora positivamente el ítem 1, este es recomendado al usuario 3 (filtrado colaborativo basado en usuarios). Por otro lado, si los ítems 1 y 3 son similares y el usuario 3 muestra interés en el ítem 3, se le recomendará el ítem 1 (filtrado colaborativo basado en ítems).

- Basado en modelo: utiliza métodos de machine learning para entrenar los datos y encontrar patrones.

Figura 2.Filtrado colaborativo basado en usuarios y basado en ítems

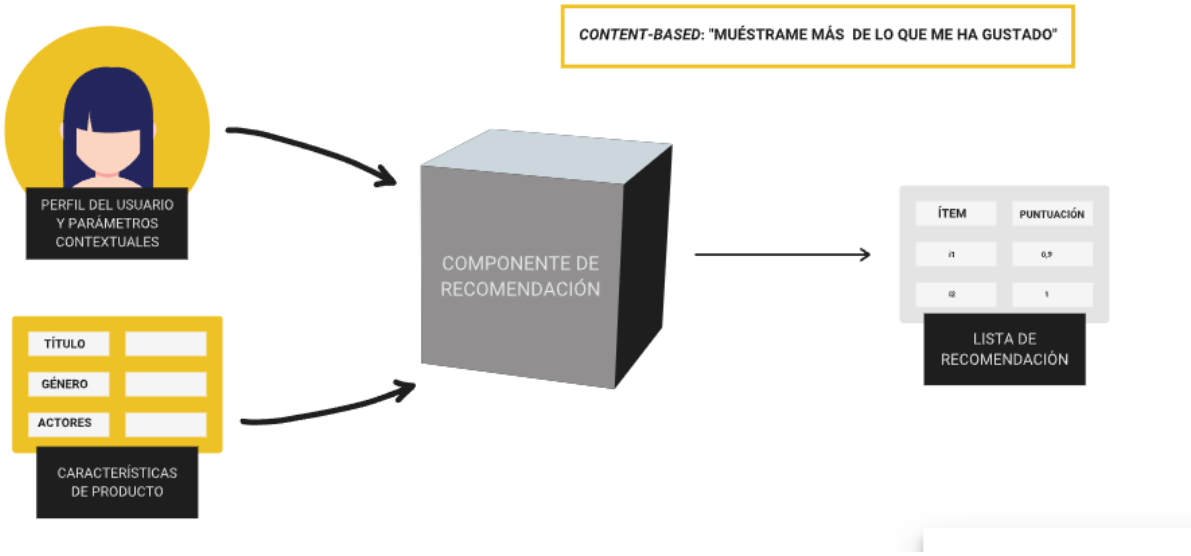


Fuente: (IBM, 2022)

2.1.2 Filtrado basado en contenido

Utiliza el perfil de las preferencias del usuario y el contenido o características del ítem. Recomienda los ítems que tienen un contenido similar a aquellos que los usuarios prefirieron en el pasado como se muestra en la Figura 3. Una de sus ventajas es que permite trabajar con nuevos ítems, pero tiene limitaciones para hacer recomendaciones novedosas o para recomendar contenido a usuarios nuevos (Lops et al., 2011).

Figura 3.Sistema recomendador basado en contenido



Fuente: (Barrenetxea, 2022)

2.1.3 Sistemas recomendadores híbridos

Combinan 2 o más métodos de filtrado para mitigar las limitaciones de cada uno y ofrecer mejores predicciones. Existen 4 subclases:

- Ponderados (“Weighted”): combina las puntuaciones de las diferentes técnicas individuales.
- Conmutados (“Switching”): prueba iterativamente cada una de las diferentes técnicas individuales hasta alcanzar la que dé mejor resultado.
- En cascada (“Cascade”): las técnicas se ordenan en secuencia para que la salida (recomendación) de una sea la entrada de la siguiente.
- Mixto (“Mixed”): presenta los resultados de varios recomendadores que trabajan datos diferentes.

2.2 Aplicaciones en diversas industrias

Una aplicación novedosa del uso de filtrado colaborativo es la predicción del sentimiento de tweets. Kim et al. (2013) comparan 4 algoritmos de filtrado colaborativo: promedio simple de ratings, filtrado colaborativo basado en usuarios, descomposición en valores singulares (SVD) y factorización matricial probabilística (PMF). La métrica utilizada fue la raíz del error cuadrático medio (RMSE) y el algoritmo PMF obtuvo los mejores resultados, demostrando la utilidad de esta técnica para predecir el sentimiento en una conversación de Twitter.

Mokarrama et al. (2020) utilizan un sistema recomendador basado en contenido para ayudar a estudiantes de Bangladesh a elegir entre diferentes universidades para postular. Para esto usan atributos relativos al desempeño académico de los postulantes y atributos relacionados a la universidad (ranking, rating de los usuarios y tarifas). En base a una recomendación binaria, clasificada como positiva o negativa, evalúan su sistema obteniendo una exactitud de 72% en la predicción.

Con respecto a los sistemas recomendadores híbridos, Gupta, J. & Gadge (2014) proponen una metodología para combinar el filtrado colaborativo basado en el rating de los usuarios con la generación de clústeres de usuarios usando datos demográficos a través de un esquema de pesos para cada técnica. Los resultados demuestran una solución escalable y que ataca el problema del "cold start".

En la misma línea, Pazzani (1999) combina un sistema recomendador basado en filtrado colaborativo con uno basado en contenido obteniendo una precisión de 70%, por encima del 68% y del 61% que obtuvo cada algoritmo por separado para dicha métrica. Además, en otro experimento agrega recomendaciones basadas en datos demográficos y combina los resultados de los 3 algoritmos logrando mejorar la precisión hasta un 72%.

2.3 Aplicaciones en el sector financiero

Si bien las aplicaciones más conocidas de sistemas recomendadores son las de contenido de música o video (Spotify, Netflix), comercio electrónico (Amazon, Ebay) y contenido de redes sociales (Facebook, Twitter), actualmente la mayoría de empresas, independientemente del rubro, están empezando a implementar estas soluciones para entender las necesidades de sus clientes y sugerirles productos o servicios (Beheshti et al., 2020).

El sector financiero no es ajeno a esta tendencia y por ello existen diversos estudios sobre la aplicación de sistemas recomendadores en este rubro. Zibriczky (2016) repasa los diferentes aportes desde dos enfoques: por dominio y por método.

Con respecto al dominio se mencionan los usos en banca para recomendar productos, ofertas de consumos en lugares cercanos o herramientas para el manejo de las finanzas personales. También se comentan varios estudios en el ámbito de préstamos, seguros, bienes inmuebles, acciones y gestión de portafolio de inversiones.

Los métodos utilizados incluyen el filtrado colaborativo (seguros, bienes inmuebles y acciones), filtrado basado en contenido (préstamos), filtrado basado en conocimiento (productos bancarios), filtrado basado en casos (seguros y portafolio de inversiones) y los métodos híbridos que combinan los previamente mencionados y que por ende son aplicables para todos los dominios.

Klioutchnikov et al. (2020) agregan en su estudio otros métodos como: filtrado demográfico, que utiliza atributos como género, edad, nivel educativo o ingresos para realizar la recomendación de un producto o servicio; filtrado basado en utilidad, para recomendar acciones o servicios en base a la utilidad que tienen para los clientes y filtrado basado en objetivos, que recomienda una serie de acciones para llegar a un estado (muy utilizado en la recomendación de portafolio de inversiones).

El alcance del presente trabajo se enfoca en la generación de ofertas comerciales en el ámbito bancario, por ello, en adelante se revisan algunos estudios previos realizados en este dominio.

Uno de los principales problemas para aplicar métodos de filtrado colaborativo para recomendar productos es que estos no suelen ser calificados por los clientes, ante esto Sharifihosseini (2019) utiliza un modelo basado en la frescura, frecuencia y el monto de las transacciones hechas con tarjetas de débito para determinar el rating de un producto. Esta propuesta se basa en un modelo de marketing para segmentar clientes llamado RFM (siglas en inglés de Recency, Frequency y Monetary) el cual analiza y predice el comportamiento de los clientes en base al comportamiento de sus transacciones (Wei et al., 2010).

Para generar recomendaciones más fiables, Gallego Vico et al. (2012) utilizan un sistema recomendador basado en el contexto de la situación, el cual dividen en 3 fases:

- Contexto social: consideran que 2 clientes son similares si tienen tendencias de compra parecidas y si también comparten algunos atributos demográficos como el género, la edad, entre otros.
- Contexto de localización: se utiliza la ubicación del dispositivo móvil del cliente para filtrar las posibles recomendaciones de lugares.
- Contexto de usuario: usa parámetros como la hora, actividad y las preferencias de los usuarios sobre la recomendación que desean recibir (restaurantes, supermercados, cines).

Cabe mencionar que su sistema incluye la posibilidad de incluir feedback cualitativo sobre la recomendación, lo que es utilizado para ajustar el modelo, en especial el contexto social y con ello dar mejores recomendaciones.

Oyebode & Orji (2020) proponen un modelo híbrido que combina el filtrado colaborativo basado en ítems con el filtrado demográfico para recomendar productos bancarios. Con ello buscan combatir los problemas clásicos en este ámbito que son la falta de ratings explícitos sobre los productos ("data sparsity") y la dificultad de recomendar un producto a un cliente nuevo ("cold start"). Para el primer caso, crean un algoritmo que genera un rating implícito de los productos en base a su frecuencia de uso y para el segundo combinan ambos métodos de filtrado a través de un sistema de pesos dinámico propuesto por los autores.

Kaya et al. (2021) construyen un sistema recomendador basado en los hábitos de consumo de los clientes con sus tarjetas de crédito. Para ello utilizan la técnica de filtrado colaborativo basado en el algoritmo de mínimos cuadrados alternos (ALS) para realizar las predicciones. Una propuesta interesante es el uso de los códigos MCC (del inglés, Merchant Category Code) que se utilizan en el sector bancario para segmentar a las empresas proveedoras de bienes y servicios (VISA Commercial Solutions, 2007). Estos códigos fueron clasificados en 16 categorías lo que les permitió asociar una campaña comercial a cada una de ellas.

Un aporte muy novedoso es la propuesta de Hernández-Nieves et al. (2020) sobre una arquitectura de "fog computing" para recomendar productos bancarios. Se basa en un nuevo paradigma que extiende los servicios cloud utilizando dispositivos de la red local de una sucursal para ejecutar un sistema recomendador híbrido que combina el filtrado colaborativo y el basado en contenido. En cuanto a las métricas de evaluación, si bien no se da mucho detalle, se propone validar las recomendaciones utilizando el razonamiento basado en casos ("Case-based reasoning").

2.4 Arquitecturas y soluciones en tiempo real

La era del big data trae consigo que los sistemas recomendadores tradicionales que capturan, analizan y actualizan los modelos en modo batch, es decir en intervalos regulares de tiempo, no puedan satisfacer las necesidades de las aplicaciones actuales, las cuales deben soportar el incremento del volumen y la velocidad con la que se generan los datos (Huang et al., 2015).

En este contexto surge la necesidad de que los sistemas recomendadores puedan tomar como entrada los datos en "stream" que vienen generando las redes sociales o el IoT y

combinarlos con los datos históricos para ofrecer recomendaciones en tiempo real (Numnonda, 2018).

Chandramouli et al. (2011) describen teóricamente algunas características que deben cumplir estos sistemas con la presentación de “StreamRec”, un sistema recomendador capaz de realizar procesamiento incremental en tiempo real en base a eventos (nuevas opiniones de usuarios) que se utilizan para actualizar el modelo, recibir solicitudes de recomendación a demanda, así como mantener la frescura de las recomendaciones al dar menor peso a las opiniones más antiguas.

A continuación, se mencionan algunos trabajos previos con énfasis en las arquitecturas y tecnologías utilizadas en el ámbito de la recomendación en tiempo real:

Tanto Numnonda (2018) como Sunny et al. (2017) proponen arquitecturas lambda basadas en 3 capas:

- Batch: usada para el procesamiento de los datos históricos y ejecución offline de modelos (Ej: clustering de clientes, filtrado colaborativo sobre los datos históricos).
- Real-time: encargada de capturar los datos en tiempo real generados por el usuario.
- Servicio: sirve de nexo entre las capas batch y real-time, combina los resultados y presenta al usuario la recomendación.

En el caso de Numnonda (2018) utiliza como persistencia HDFS de Apache Hadoop (Apache Software Foundation, 2006) y Apache Spark (Zaharia et al., 2016) para el procesamiento en la capa batch. En la capa real-time usa Apache Kafka (Apache Software Foundation, 2014a) para el manejo de los “streams” de datos y Spark Streaming para realizar la inferencia con el modelo recomendador construido en Spark MLlib.

Por su parte, la solución de Sunny et al. (2017) se basa en el ecosistema de Apache Spark tanto para el procesamiento batch, como para el de tiempo real con Spark Streaming y la implementación de modelos con Spark MLlib. Aunque no menciona específicamente la tecnología con la cual captura los “streams” de datos, aporta una comparativa de 3 bases de datos NoSQL: Apache Cassandra (Apache Software Foundation, 2008a), CouchDB (Apache Software Foundation, 2008b) y MongoDB (MongoDB Inc, 2009) para persistir los datos generados en ambas capas, siendo elegida la primera al obtener menor latencia en las pruebas.

Singhal et al. (2019) detallan el desarrollo de iPrescribe, una arquitectura escalable y de baja latencia para recomendar la siguiente mejor oferta de manera online. Se mencionan 5 capas

de la arquitectura: mensajería con Apache Kafka, almacenamiento persistente en HDFS, almacenamiento en memoria, procesamiento en tiempo real, ambos con Apache Ignite (Apache Software Foundation, 2015) y Tornado (Darnell & Taylor, 2009) como framework para las aplicaciones web. Adicionalmente, se menciona el uso de Python (Van Rossum, 1995) para el desarrollo de los modelos.

Un sistema recomendador de propósito general fue construido por la compañía china Tencent que afirma estar en producción procesando 10 billones de transacciones y más de un 1 petabyte de datos diariamente, además de dar acertadas recomendaciones en tiempo real (Huang et al., 2015). Su arquitectura consiste en un clúster de miles de máquinas que realizan el procesamiento en tiempo real a través de Apache Storm (Apache Software Foundation, 2014b). Una novedad de su arquitectura, es el desarrollo "in-house" de soluciones para el acceso a los datos (TDAccess) y para la persistencia de datos en memoria (TDStore).

2.5 Conclusiones

Como se ha podido revisar, existen numerosos estudios sobre sistemas recomendadores debido a su amplio espectro de aplicación. Por ello, el presente trabajo toma de referencia alguno de los aportes realizados en el ámbito bancario para desarrollar un sistema recomendador en tiempo real que ayude a los equipos comerciales en la generación de ofertas personalizadas a los clientes.

La propuesta implica el desarrollo de un algoritmo que pueda predecir el rating que le dará un cliente a un producto para en base a ello realizar la recomendación e iniciar la gestión de la oferta comercial. Ya que no es usual contar con los ratings de los productos, se toma el enfoque de Oyebode & Orji (2020) de generar ratings implícitos para luego aplicar un algoritmo de filtrado colaborativo basado en ítems y combinarlo con un clustering basado en los datos de los clientes para combatir el problema del "cold start".

El mencionado algoritmo solo considera el tiempo transcurrido desde el último uso de un producto para determinar un rating implícito que va del 1 (más antiguo) al 5 (más reciente). La contribución en este sentido consiste en incorporar las variables utilizadas en el modelo RFM: frescura, frecuencia y monto de las transacciones por cada producto para determinar el rating de manera similar a lo realizado por Sharifihosseini (2019) en su estudio sobre un sistema recomendador basado en la transaccionalidad de tarjetas.

Con respecto a los productos a recomendar, se toma la propuesta de Kaya et al. (2021) de categorizar los códigos MCC que se registran en cada transacción hecha con tarjeta para agruparlos en categorías según los hábitos de consumo de los clientes. El objetivo de esto

es poder generar las ofertas personalizadas en base a categorías de consumo similares a las que el cliente prefirió en el pasado.

Otro de los aportes de la propuesta es la posibilidad de dar recomendaciones en tiempo real, por lo cual se mencionaron algunos estudios sobre sistemas recomendadores con esta característica. Para ello, es necesario implementar una arquitectura que combine el procesamiento batch con el de tiempo real tal como proponen Numnonda (2018) y Sunny et al. (2017) utilizando tecnologías bastante maduras en este aspecto como Apache Kafka para la gestión de los mensajes en “streaming”, Apache Spark tanto para el procesamiento de los datos en batch como en tiempo real y Apache Hadoop para almacenar los datos de entrada y los resultados obtenidos.

3. Objetivos concretos y metodología de trabajo

3.1 Objetivo general

El objetivo del presente trabajo es diseñar un sistema recomendador que trabaje en tiempo real y que sirva a los equipos comerciales en la gestión de ofertas personalizadas para los clientes. Para lograr esto, el sistema recibe como entrada la transaccionalidad de las tarjetas de crédito o débito de los clientes, además de atributos demográficos y otras variables disponibles, con las cuales debe predecir el rating que le dará un cliente a una categoría de consumo definida y en base a ello realizar una recomendación.

3.2 Objetivos específicos

En base al objetivo general planteado, se definen los siguientes objetivos específicos:

- Diseñar una arquitectura que pueda capturar las transacciones que hace un cliente con su tarjeta en tiempo real.
- Implementar un algoritmo que calcule los ratings implícitos para cada cliente y categoría de consumo en base a la frescura, frecuencia y monto de las transacciones.
- Construir un modelo de machine learning basado en filtrado colaborativo que permita predecir el rating que le dará un cliente a una categoría específica. Este modelo se utilizará para clientes ya existentes.
- Construir un modelo de clustering con los atributos de los clientes para segmentarlos y asignar un rating promedio para cada clúster y categoría de consumo. Este modelo se utilizará cuando un cliente es nuevo.
- Procesar los datos conforme van llegando y enviarlos al sistema recomendador para realizar la predicción en tiempo real.
- Evaluar el desempeño de los modelos con las métricas del error absoluto medio (MAE) y la raíz del error cuadrático medio (RMSE). Para evaluar la calidad de los clústeres generados se usará la métrica silhouette.
- Almacenar los resultados de las recomendaciones para calibrar el modelo o para realizar futuros análisis con esos datos.
- Hacer disponible la salida del modelo para el consumo por parte de los aplicativos de gestión comercial y usuarios finales.

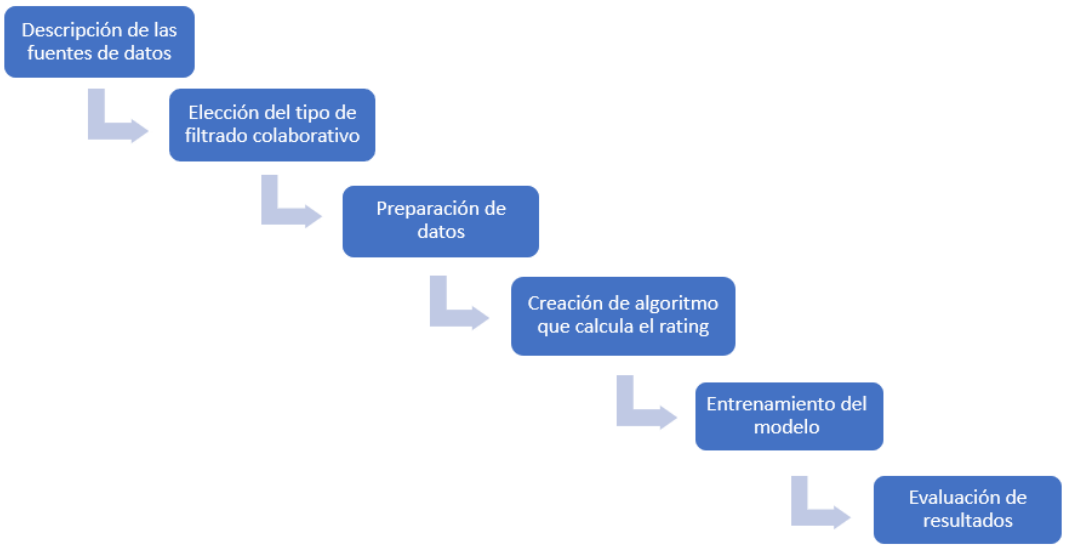
3.3 Metodología del trabajo

Para lograr los objetivos propuestos se divide el trabajo en 3 etapas: diseño de un sistema recomendador basado en filtrado colaborativo, diseño del modelo de clustering con atributos del cliente y la propuesta de una arquitectura que permita realizar las recomendaciones en tiempo real. Cada una de las etapas cuenta con una serie de actividades que se detallan a continuación:

3.3.1 Modelo basado en filtrado colaborativo

El objetivo de esta etapa es diseñar una solución de filtrado colaborativo en base a los datos de transacciones con tarjetas de crédito y débito que permita predecir el rating que le dará un cliente a una categoría de consumo para en base a ello generar una recomendación personalizada. En la Figura 4 se detallan las actividades correspondientes a esta etapa:

Figura 4. Metodología para el modelo de filtrado colaborativo

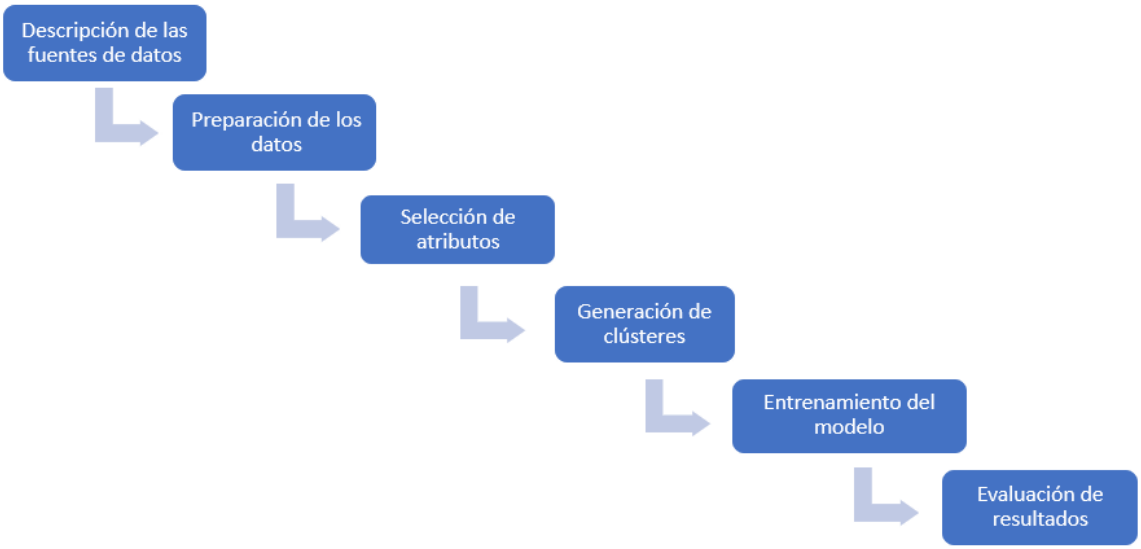


Fuente: Elaboración propia

3.3.2 Modelo basado en clustering

Como se menciona en diversos estudios revisados en el capítulo 2, el principal problema de solo aplicar filtrado colaborativo es el “cold start”, es decir, la incapacidad del sistema de dar una recomendación a un cliente nuevo. Para estos casos se propone un sistema recomendador basado en un algoritmo de clustering para agrupar a los clientes según sus atributos demográficos además de otras variables que se disponen. En la Figura 5 se detallan los pasos a seguir en esta etapa.

Figura 5. Metodología para el modelo de clustering

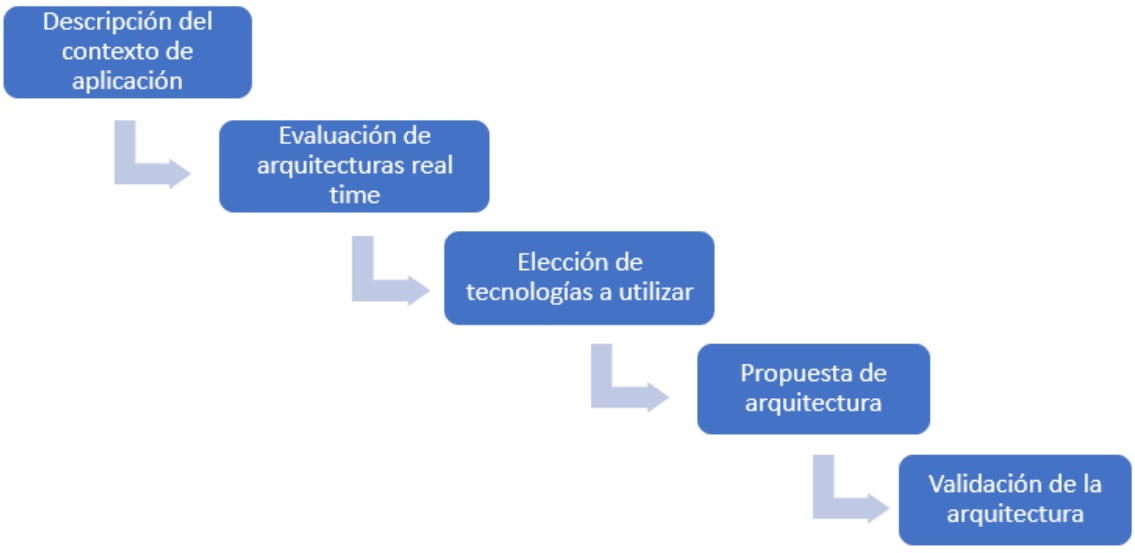


Fuente: Elaboración propia

3.3.3 Arquitectura real time

Uno de los principales aportes del trabajo es diseñar una arquitectura que pueda capturar los datos de las transacciones en tiempo real y enviarlas al sistema de recomendación para realizar las predicciones. Con estos resultados, se podrá iniciar la gestión y el envío oportuno de las ofertas comerciales a los clientes. Para esta etapa se consideran las actividades de la Figura 6.

Figura 6. Metodología para el diseño de la arquitectura real time



Fuente: Elaboración propia

4. Desarrollo del sistema recomendador

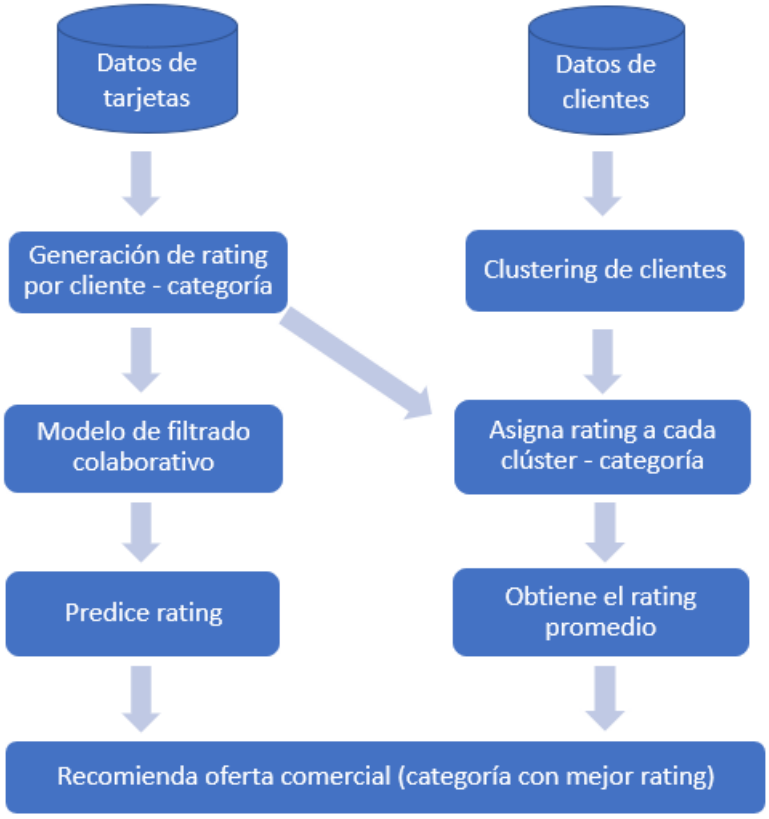
Este capítulo aborda todo lo relacionado al diseño e implementación del sistema recomendador. Como se mencionó en el capítulo anterior, se tienen 2 modelos que se utilizarán dependiendo si el cliente ya ha realizado consumos en las categorías seleccionadas para el estudio o no.

Si el cliente ya ha realizado algún consumo en al menos una de las categorías, se utilizará el modelo basado en filtrado colaborativo para realizar la predicción del rating para el resto de categorías y se le hará una oferta comercial relacionada a la que tenga el mejor rating.

Por el contrario, si el cliente es nuevo o no ha consumido en ninguna de las categorías elegidas, se utilizará un modelo de clustering basado en los atributos demográficos y demás variables para clasificar a los clientes en *k* clústeres. Al llegar un nuevo cliente, se le asignará al clúster más cercano y el rating será determinado por el promedio de calificaciones de los clientes de ese clúster para cada categoría. De igual modo, la oferta comercial estará definida por la categoría con mejor rating en dicho clúster.

En la Figura 7 se detalla a alto nivel el funcionamiento del sistema recomendador descrito.

Figura 7.Funcionamiento del sistema recomendador propuesto



Fuente: Elaboración propia

4.1 Modelo basado en filtrado colaborativo

4.1.1 Descripción de las fuentes de datos

Como se ha mencionado previamente, la propuesta toma como insumo las transacciones hechas por los clientes con sus tarjetas de crédito o débito. En el sistema transaccional del banco, sobre el cual se está realizando este estudio, se cuenta con 2 archivos con el detalle de los movimientos, uno para cada tipo de tarjeta. La estructura es la misma y los campos más relevantes se detallan en la Tabla 1.

Tabla 1.Estructura de las tablas de movimientos de tarjeta

Campo	**Descripción**
Id_Movimiento	Código correlativo que identifica a la transacción.
Id_Contrato	Hace referencia al contrato de la entidad con el cliente para el producto tarjeta de crédito o débito.
Cod_MCC	Código que identifica el tipo de comercio donde se está realizando la transacción.
Mnt_Transacción	Monto de la transacción en moneda local.
Desc_Transaccion	Información adicional sobre la transacción
Fec_Contable	Corresponde a la fecha de la transacción.

Fuente: Elaboración propia

El campo Cod_MCC hace referencia al código de cuatro dígitos conocido como MCC mencionado en el capítulo 2. La estructura de esta tabla maestra con todas las descripciones de los códigos se muestra en la Tabla 2.

Tabla 2.Estructura de la tabla MCC

Campo	**Descripción**
Cod_MCC	Código de 4 dígitos que identifica el tipo de comercio.
Desc_MCC	Descripción del tipo de comercio.

Fuente: Elaboración propia

Para poder recuperar el código del cliente a quien le corresponde la tarjeta debemos usar una tabla que relaciona al cliente con el contrato respectivo. Esta estructura se detalla en la Tabla 3 que se muestra a continuación.

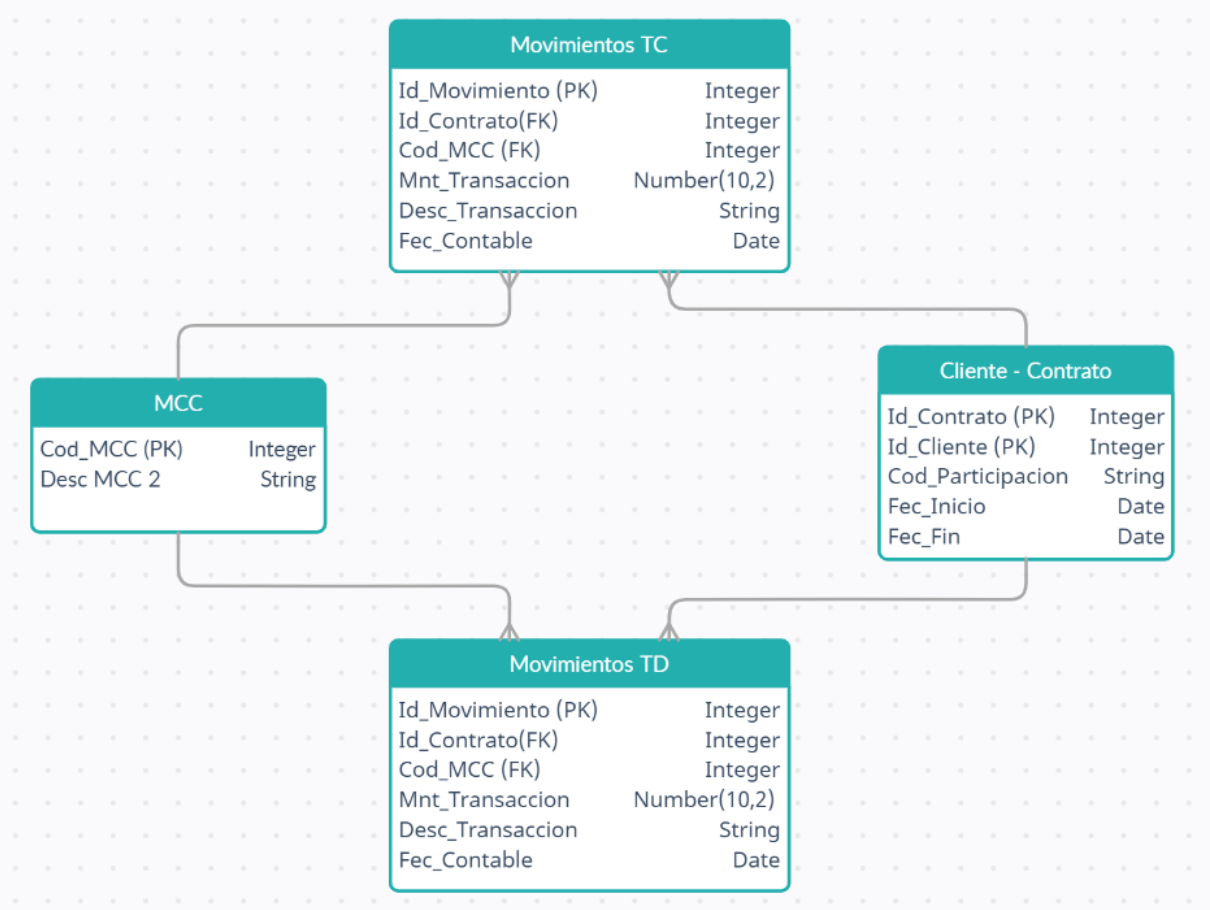
Tabla 3.Estructura de la tabla que relaciona los clientes con los contratos

Campo	**Descripción**
Id_Cliente	Código interno de la entidad para identificar a sus clientes.
Id_Contrato	Código del contrato de la entidad con el cliente para un producto específico.
Cod_Participacion	Indica el tipo de participación en la relación cliente – contrato. Ejemplo: Titular, Cónyuge, Dependiente
Fec_Inicio	Fecha inicio de la relación contractual
Fec_Fin	Fecha fin de la relación contractual

Fuente: Elaboración propia

En la Figura 8 se muestra el modelo entidad – relación donde se visualizan más claramente las relaciones de las tablas previamente descritas.

Figura 8.Modelo entidad - relación de las fuentes de datos



Fuente: Elaboración propia

4.1.2 Elección del tipo de filtrado colaborativo

Como se describió en el estado del arte, los sistemas de filtrado colaborativo basados en memoria pueden ser de 2 tipos dependiendo de si se utiliza la similitud entre usuarios o ítems. En su estudio sobre sistemas recomendadores basados en filtrado colaborativo, Sarwar et al. (2001) afirman que, para realizar la predicción, estos algoritmos parten de una matriz de ratings de *i* usuarios por *j* ítems donde cada valor de la matriz corresponde al rating que le otorga el usuario *i* al ítem *j*.

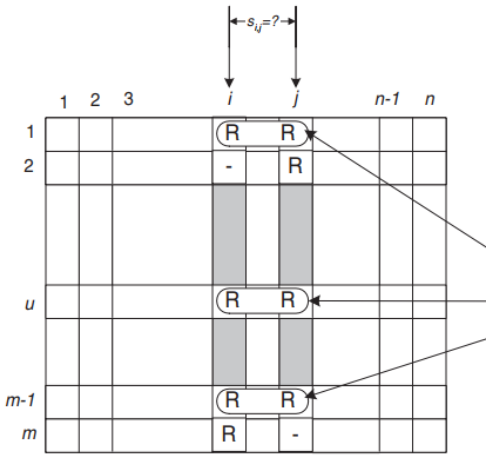
También mencionan que el principal problema de calcular la similitud entre usuarios es que al recibir una nueva petición de recomendación debe calcularse la similitud del cliente en cuestión contra el resto de clientes. Esto en entornos con grandes volúmenes de datos o cuyo requisito sea dar respuestas rápidas puede derivar en serios problemas de rendimiento.

Debido a que la solución que se plantea en el presente trabajo cumple ambas características: procesa datos masivos ya que utiliza las transacciones de tarjetas y además requiere ofrecer recomendaciones en tiempo real, la mejor alternativa es utilizar la similitud entre ítems.

Por otro lado, la ventaja de los ítems es que su cantidad suele ser menor al número de usuarios y aumentan en menor medida que estos. Además, la similitud entre ítems no requiere ser calculada cada vez que se solicita una recomendación, sino que puede ser calculada de antemano lo que proporciona respuestas más rápidas ideales para entornos en tiempo real como el planteado en esta propuesta.

El cálculo de la similitud entre ítems se realiza solo considerando los usuarios que han calificado ambos ítems (Figura 9) utilizando alguno de los métodos que Sarwar et al. (2001) definen a continuación:

Figura 9.Cálculo de la similitud entre los ítems i y j



Fuente: (Sarwar et al., 2001)

- Similitud de coseno: los ítems son considerados como vectores y la similitud se define como el coseno del ángulo formado por dichos vectores tal como se muestra en la ecuación 1.

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\left|\left|\vec{i}\right|\right|^2 * \left|\left|\vec{j}\right|\right|^2} \quad (1)$$

- Similitud basada en la correlación: la ecuación 2 muestra la fórmula para calcular la similitud entre 2 ítems utilizando a correlación de Pearson, donde $R_{u,i}$ representa el rating del usuario *u* sobre el ítem *i* y $\overline{R_i}$ es el rating promedio del ítem *i*.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}} \quad (2)$$

- Similitud de coseno ajustado: resta la media de los ratings de un usuario *u* ($\overline{R_u}$) para contrarrestar el efecto de la diferencia de escalas al momento de calificar un ítem. La fórmula para este caso se muestra en la ecuación 3.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}} \quad (3)$$

4.1.3 Preparación de los datos

De acuerdo a lo descrito en el punto anterior, para calcular la similitud entre ítems se necesita construir una tabla que contenga la relación entre el cliente, el producto y el rating correspondiente. Los pasos para la preparación de esta base se detallan a continuación:

- Se consideran todas las transacciones realizadas desde enero de 2020 hasta abril del 2022 tanto con tarjeta de crédito como de débito obteniendo una base preliminar de más de 400 millones de registros.

- Se cruza contra la tabla que contiene la relación entre cliente y contrato para recuperar el código del cliente. Se consideran solo los contratos activos a la fecha de la consulta y en los cuales el cliente sea el titular.

- Luego, se identifican los códigos MCC que representan el 80% del total de movimientos, obteniéndose un total de 14 que se agrupan en 9 categorías de consumo descritas en la Tabla 4. Estas categorías corresponden a los ítems (productos) sobre los cuales se va a realizar la predicción.

- Hasta este punto, la base cuenta con más de 240 millones de registros correspondientes a transacciones de más de 3 millones de clientes. Para filtrar los casos de clientes que han hecho muy pocas transacciones a lo largo de los 2 años, se filtran solo aquellos clientes con más de 500 transacciones.

- Finalmente, para poder calcular la frescura de las transacciones, siguiendo el enfoque de Sharifihosseini (2019) para calcular el rating implícito, se crea un campo con la diferencia entre la fecha actual (para efectos de la prueba se considera el 30/04/2022) y la fecha de la transacción. La base final contiene más de 63 millones

de registros correspondientes a 78 211 clientes y una muestra puede visualizarse en la Figura 10.

Tabla 4.Categorías de consumo según códigos MCC

Categoría	**Códigos y Descripción MCC**
Alimentación	5411 – Supermercados y Minimarkets 5499 – Bodegas y Pastas 5462 – Panaderías/Pastelerías
Operaciones Bancarias	6011 – Bancos, Pagos Automáticos 6012 – Instituciones Financieras
Restaurantes	5812 – Restaurantes 5814 – Servicios Express / Snacks / Fast Foods
Grifos	5541 – Grifos
Farmacias	5912 – Farmacias
Transporte	4121 – Servicios de Taxi
Servicio de delivery	4789 – Servicios de transporte no clasificados
Tiendas por departamento	5311 – Tiendas por departamento
Servicios	4814 – Servicios de Telecomunicaciones 4899 – Servicios de pago (TV y Radio)

Fuente: Elaboración propia

Figura 10. Muestra de la tabla resultante luego de la preparación de datos

Id_cliente	Categoria	Monto	Frescura
24673321	Operaciones Bancarias	400.00	1
24673321	Servicios de Delivery	50.80	1
24673321	Alimentacion	126.12	4
24673321	Tiendas por Departamento	79.98	5
24673321	Operaciones Bancarias	400.00	8
24673321	Alimentacion	20.94	9
24673321	Servicios de Cable	1169.00	10
24673321	Servicios de Cable	18.90	11
24673321	Alimentacion	103.26	11
24673321	Transporte	8.60	11

Fuente: Elaboración propia

4.1.4 Algoritmo para cálculo de rating implícito

Choi et al. (2012) afirman que lo más común es no contar con la información de los ratings de los productos por lo cual es importante un nuevo enfoque para derivar los ratings de la información transaccional y poder aplicar la técnica de filtrado colaborativo. Si bien esta

afirmación se hizo para el ámbito del comercio electrónico, se aplica perfectamente en el sector bancario donde los clientes no suelen calificar los productos que tienen.

En su estudio, Choi et al. (2012) utilizan la frecuencia de compra para calcular el rating implícito utilizando la fórmula de la ecuación 4 donde *RP(u,i)* define la preferencia relativa del usuario *u* sobre el ítem *i* en base a la preferencia absoluta *AP(u,i)* del usuario *u* sobre el ítem *i.* El resultado se divide entre el máximo *AP(u,i)* existente por cada ítem para obtener un rating entre 0 y 1 al cual se le aplica la ecuación 5 para establecer el valor en una escala de 1 a 5 que es la más utilizada a la hora de calificar un ítem.

$$RP(u,i) = \frac{AP(u,i)}{MAX_{c \in U}(AP(c,i)} \tag{4}$$

$$Donde: AP(u,i) = \left(\frac{Nro\ de\ transacciones\ del\ usuario\ u\ que\ incluyen\ el\ ítem\ i}{Total\ de\ transacciones\ del\ usuario\ u}\right) + 1$$

$$Rating\ Final(u,i) = Redondear(5 * RP(u,i)) \tag{5}$$

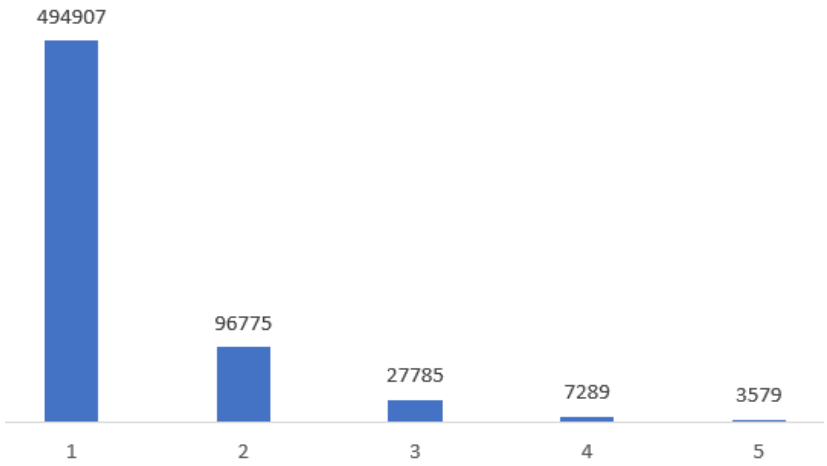
Al aplicar este método con los datos disponibles, se notó que existían clientes cuyas transacciones pertenecían en su gran mayoría a una sola categoría y por lo tanto generaban que el máximo *AP(u,i)* fuera cercano a 1, como se muestra en la Figura 11, lo cual tuvo un efecto en la distribución de los ratings concentrándolos en los valores más bajos como se ve en la Figura 12.

Figura 11.Cálculo de la preferencia absoluta máxima para cada categoría

Categoria	max_AP
Grifos	0.9991617770326907
Servicios de Cable	1.0
Operaciones Bancarias	1.0
Restaurantes	0.9984825493171472
Transporte	0.9984276729559748
Servicios de Delivery	0.998330550918197
Farmacias	0.9949109414758269
Alimentacion	0.9987819732034104
Tiendas por Departamento	1.0

Fuente: Elaboración propia

Figura 12.Distribución de ratings calculados con el algoritmo de Choi et al. (2012)



Fuente: Elaboración propia

Debido a que el estudio de Oyebode & Orji (2020) también se basa en una única variable, en ese caso la frescura de las transacciones, se decidió probar el enfoque de Sharifihosseini (2019) de utilizar las tres variables del modelo RFM: frescura, frecuencia y monto de las transacciones. El algoritmo utilizado se describe a continuación:

- Se agrupan los datos por cliente y categoría para obtener las 3 variables para cada grupo. Para la frecuencia se contabilizan las transacciones, en el caso de la frescura se toma el mínimo valor del campo con el mismo nombre (Figura 10) y para el monto se suman el valor de las transacciones.

- Luego, para cada cliente se ordenan las variables de frecuencia y monto de mayor a menor para establecer un ranking para cada registro empezando desde 1.

- Calculados esos valores, se procede a asignar un rating para cada variable siguiendo la lógica de la Tabla 5.

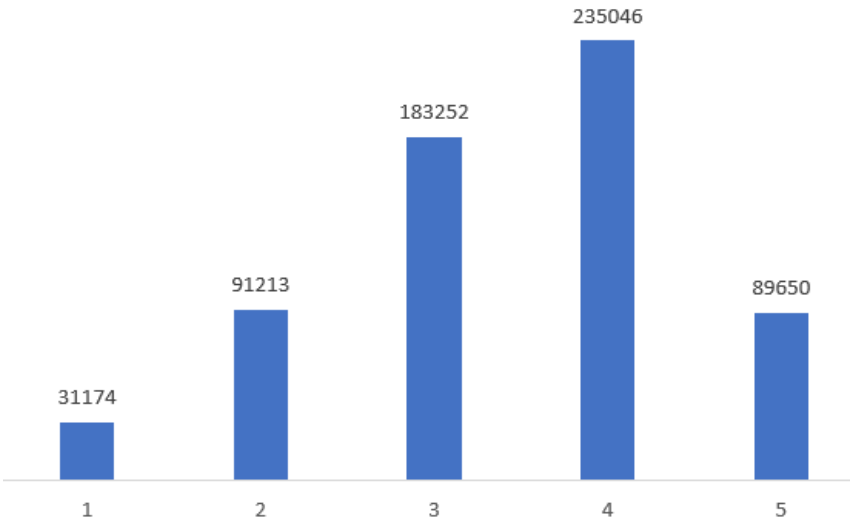
- Finalmente, se calcula el rating final a través de un promedio ponderado de los ratings obtenidos para cada variable por cliente y categoría. Los pesos elegidos fueron: 40% para la frecuencia, 30% para la frescura y 30% para el monto. Se observa una distribución más balanceada como se ve en la Figura 13.

Tabla 5.Lógica para calcular los ratings de cada variable de RFM

Variable	**Lógica**
Frescura	$\text{Rating} = \begin{cases} 5 & \text{si Frescura} \leq 60 \\ 4 & \text{si Frescura} \in [61, 120] \\ 3 & \text{si Frescura} \in [121, 180] \\ 2 & \text{si Frescura} \in [181, 360] \\ 1 & \text{si Frescura} > 360 \end{cases}$
Frecuencia	$\text{Rating} = \begin{cases} 5 & \text{si rankingFrecuencia} = 1 \\ 4 & \text{si rankingFrecuencia} \in [2,3] \\ 3 & \text{si rankingFrecuencia} \in [4,5] \\ 2 & \text{si rankingFrecuencia} \in [6,7] \\ 1 & \text{si rankingFrecuencia} \in [8,9] \end{cases}$
Monto	$\text{Rating} = \begin{cases} 5 & \text{si rankingMonto} = 1 \\ 4 & \text{si rankingMonto} \in [2,3] \\ 3 & \text{si rankingMonto} \in [4,5] \\ 2 & \text{si rankingMonto} \in [6,7] \\ 1 & \text{si rankingMonto} \in [8,9] \end{cases}$

Fuente: Elaboración propia

Figura 13.Distribución de ratings con algoritmo basado en RFM



Fuente: Elaboración propia

Una muestra del conjunto de datos final que se utilizará para el entrenamiento del modelo de filtrado colaborativo se puede visualizar en la Figura 14.

Figura 14.Muestra de la base final a utilizar para entrenar el modelo

Id_Cliente	Categoria	Rating_ponderado
24673321	Tiendas por Departamento	4.0
24673321	Operaciones Bancarias	4.0
24673321	Alimentacion	4.0
24673321	Transporte	4.0
24673321	Restaurantes	4.0
24673321	Servicios de Cable	3.0
24673321	Farmacias	2.0
24673321	Servicios de Delivery	2.0
24673321	Grifos	1.0

Fuente: Elaboración propia

4.1.5 Entrenamiento del modelo

Para esta tarea se hace uso de la librería Surprise la cual proporciona un conjunto de herramientas para construir sistemas recomendadores en Python (Hug, 2019).

Surprise cuenta con una serie de datasets muy conocidos en este ámbito, como es el caso de MovieLens (GroupLens, 2019), pero también permite trabajar con datasets propios mientras contengan las 3 variables principales: cliente, producto y rating.

En el caso de los algoritmos de filtrado colaborativo, la librería se encarga internamente de crear la matriz de cliente – producto y calcular las similitudes entre usuarios o ítems según sea el caso. Luego, aplica la fórmula para predecir el rating de acuerdo a la lógica del algoritmo que se haya escogido.

Adicionalmente, proporciona un módulo que permite gestionar el dataset para el entrenamiento y prueba a través de la técnica de validación cruzada o la división del dataset indicando un porcentaje para el entrenamiento y el restante para pruebas. El módulo de validación proporciona las métricas más utilizadas en este ámbito como el error absoluto medio (MAE), el error cuadrático medio (MSE) y la raíz del error cuadrático medio (RMSE).

Para el presente trabajo se utilizan los algoritmos de filtrado colaborativo basados en la similitud de los ítems entre los cuales tenemos:

KNNBasic: utiliza el algoritmo básico de filtrado colaborativo, que realiza la predicción del rating de un usuario *u* hacia un ítem *i* utilizando los ratings de los ítems similares a *i* ponderados según la similaridad entre los ítems *i y j* de acuerdo a la fórmula de la ecuación 6.

$$r_{ui} = \frac{\sum_{j \in N_u^k\,(i)} sim(i,j) * r_{uj}}{\sum_{j \in N_u^k\,(i)} sim(i,j)} \quad (6)$$

KNNWithMeans: difiere del anterior al considerar el rating promedio para los ítems según la fórmula de la ecuación 7.

$$r_{ui} = \mu_i + \frac{\sum_{j \in N_u^k\,(i)} sim(i,j) * (r_{uj} - \mu_j)}{\sum_{j \in N_u^k\,(i)} sim(i,j)} \quad (7)$$

KNNWithZScore: realiza una normalización del rating de cada usuario como se ve en la ecuación 8.

$$r_{ui} = \mu_i + \sigma_i \frac{\sum_{j \in N_u^k\,(i)} sim(i,j) * \frac{(r_{uj} - \mu_j)}{\sigma_j}}{\sum_{j \in N_u^k\,(i)} sim(i,j)} \quad (8)$$

Para evaluar qué tan bueno es el modelo propuesto, se comparará contra un clasificador que genera un rating aleatorio de acuerdo a la distribución de los datos que se asume que siguen una normal. Asimismo, se hará una comparación entre el enfoque de Oyebode & Orji (2020) que utiliza solo la frescura y la propuesta del presente trabajo que considera las 3 variables del modelo RFM: frescura, frecuencia y monto.

Por otro lado, también se evaluará el desempeño variando la medida de similitud entre ítems para lo cual hay 2 opciones: similitud de coseno o coeficiente de correlación de Pearson.

4.1.6 Discusión de resultados

Para evaluar el modelo se utiliza el módulo de validación cruzada, descrito en el apartado anterior, considerando 2 parámetros: número de folds, que se dejó en su valor por defecto (5), y las métricas escogidas, que fueron RMSE y MAE. Para las últimas se toma el promedio de las 5 iteraciones de la validación cruzada cuyos resultados se resumen en las tablas 6 y 7 que se muestran a continuación.

Tabla 6. Resultados del modelo de filtrado colaborativo usando el enfoque de Oyebode & Orji (2020)

Algoritmo	Similitud de Coseno		Coeficiente de correlación	
	RMSE	MAE	RMSE	MAE
Normal Predictor	1.5418	1.1253	1.5418	1.1253
KNNBasic	1.2362	0.8514	1.2024	0.7831
KNNWithMeans	1.1513	0.7994	1.1204	0.7665
KNNWithZScore	1.1580	0.7737	1.1411	0.7458

Fuente: Elaboración propia

Tabla 7. Resultados del modelo de filtrado colaborativo propuesto

Algoritmo	Similitud de Coseno		Coeficiente de correlación	
	RMSE	MAE	RMSE	MAE
Normal Predictor	1.4409	1.1615	1.4409	1.1615
KNNBasic	1.1854	0.9748	1.3562	1.1122
KNNWithMeans	0.9865	0.7772	1.0013	0.7791
KNNWithZScore	0.9983	0.7770	1.0126	0.7782

Fuente: Elaboración propia

Como se puede apreciar, para todos los algoritmos ejecutados, el enfoque propuesto de utilizar un rating ponderado con las 3 variables del modelo RFM obtiene mejores resultados que solo utilizar la frescura. Tomando en cuenta los mejores valores del RMSE tanto del modelo propuesto (0.9865) y el que usa solo la frescura (1.1204) se logra una mejora cercana al 12%.

También se valida que los algoritmos basados en filtrado colaborativo son significativamente mejores al predictor aleatorio que obtuvo un RMSE de 1.44 para el modelo propuesto. De hecho, el algoritmo clásico de filtrado colaborativo (KNNBasic) lo supera largamente con un promedio de RMSE de 1.1854 y dentro de este tipo de algoritmos el que considera la media de los ratings (KNNWithMeans) es el que obtiene el mejor resultado con un RMSE de 0.9865.

Asimismo, se comprueba que los mejores resultados son obtenidos al utilizar la similitud de coseno sobre el coeficiente de correlación de Pearson al momento de calcular la similitud entre los ítems.

4.2 Modelo basado en clustering

Las técnicas de clustering forman parte del aprendizaje no supervisado, es decir, cuando los datos no tienen una etiqueta o clase definida. El objetivo principal de este tipo de técnicas es encontrar agrupaciones de objetos que sean similares entre sí dentro de un grupo y que a su vez sean diferentes de los objetos que pertenecen a otros grupos (Popat & Emmanuel, 2014).

Entre las numerosas técnicas de clustering se encuentran las de agrupamientos exclusivo que se basa en un enfoque iterativo para encontrar similitud entre los puntos de un clúster en base a la distancia hacia el punto central del clúster o centroide (Ghosal et al., 2020). El algoritmo más conocido de este tipo es el K-Means que será utilizado en el presente trabajo debido a su simplicidad y al gran número de referencias encontradas sobre el uso de dicho algoritmo en sistemas recomendadores.

Los pasos del algoritmo de K-Means se detallan a continuación:

- Se elige un número *k* de clústeres a formar y sus respectivos centroides.
- Para cada punto se calculan todas las distancias hacia los centroides y se asigna dicho punto al clúster con el centroide más cercano.
- Una vez asignados todos los puntos se recalcula el valor de los centroides.
- El algoritmo finaliza cuando el valor de los centroides permanece constante luego del cálculo anterior.

4.2.1 Descripción de las fuentes de datos

Como se indicó anteriormente, para atacar el problema del “cold start” se propone utilizar un modelo de clustering con los datos demográficos de los clientes, así como los montos de sus ingresos y deudas. Las variables que se disponen y algunas otras que fueron calculadas para el universo de 78 211 clientes son descritas en la Tabla 8.

Tabla 8.Atributos disponibles para modelo de clustering

Campo	**Tipo**	**Descripción**
Id_Cliente	Dato origen	Código interno de la entidad para identificar a sus clientes.
Edad	Dato origen	Informa la edad del cliente.
Género	Dato origen	Indicador del género de la persona. Posibles valores: M (Masculino) o F (Femenino)
Ingresos	Dato calculado	Monto promedio mensual de ingresos del cliente desde enero 2020.
Estado Civil	Dato origen	Indicador de la situación civil del cliente. Posibles valores: soltero, casado, viudo, divorciado, otros.
Profesión	Dato origen	Código interno de la profesión registrada por el cliente.
Ubicación	Dato calculado	Región donde vive el cliente, se deriva del territorio al que pertenece la sucursal bancaria que tiene asignada. Posibles valores: Lima, sur, norte, centro, oriente y otros.
Deuda_SF	Dato calculado	Monto promedio mensual de deuda del cliente en el sistema financiero desde enero 2020.
Anios_antiguedad	Dato calculado	Tiempo (en años) desde que el cliente inició relación con la entidad bancaria.
Ctd_Productos	Dato calculado	Cantidad de productos contratados por el cliente. Se consideran los siguientes: • Pasivos: cuenta de ahorro, CTS, plazo fijo y fondos mutuos. • Activos: préstamo personal, préstamo vehicular y préstamo hipotecario. • Seguros: vehicular, hipotecario y de tarjeta de crédito.

Fuente: Elaboración propia

4.2.2 Preparación de los datos

Como se describe en la Tabla 8, aparte del código de cliente se cuenta con 9 variables para realizar el clustering. Luego de eliminar los registros con valores nulos, el dataset resultante cuenta con 73 493 registros.

En la Figura 15 se visualizan las medidas de tendencia central y los cuartiles de las variables numéricas: edad, ingresos, deuda en el sistema financiero, años de antigüedad del cliente y cantidad de productos.

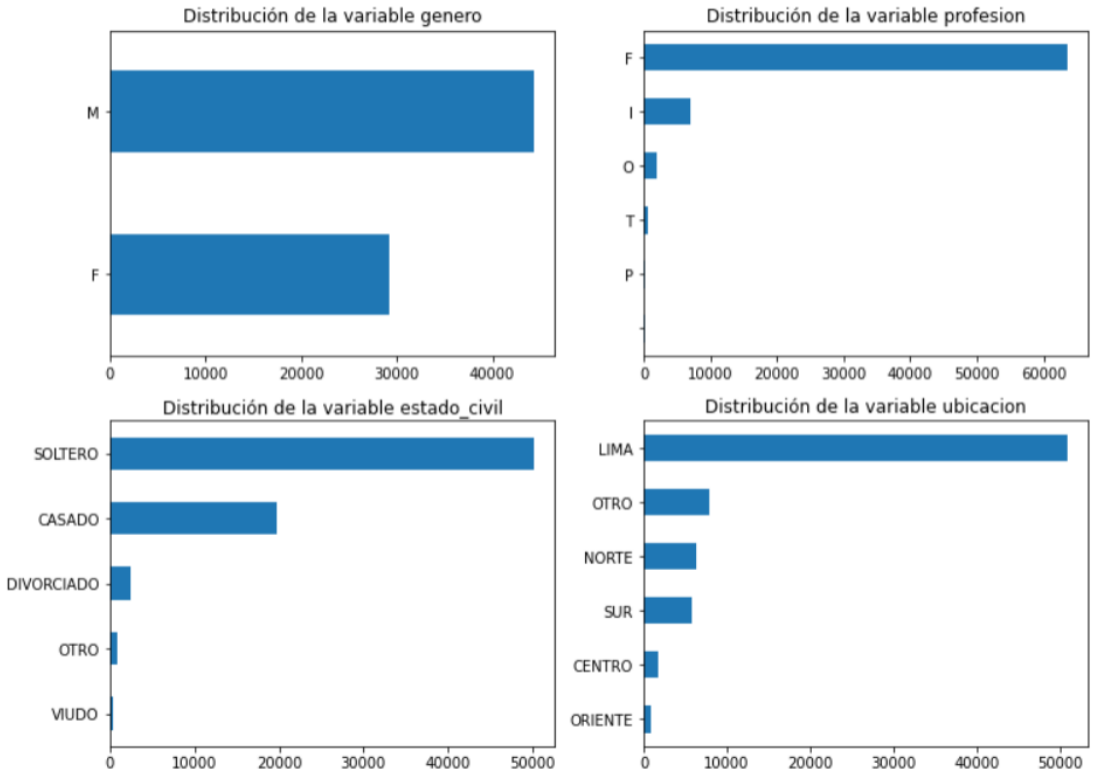
Figura 15.Principales estadísticos de las variables numéricas

	edad	ingresos	deuda_SF	anios_antiguedad	ctd_productos
count	73493.000000	73493.000000	7.349300e+04	73493.000000	73493.000000
mean	39.760671	5701.132466	8.114401e+04	12.961819	2.050835
std	11.076602	6546.258650	3.304890e+05	7.462804	1.049035
min	19.000000	49.924286	0.000000e+00	0.000000	0.000000
25%	31.000000	2673.437143	2.063660e+03	7.000000	1.000000
50%	37.000000	4064.141071	1.121182e+04	11.000000	2.000000
75%	46.000000	6629.327500	5.312616e+04	18.000000	3.000000
max	100.000000	368206.738214	3.104110e+07	54.000000	8.000000

Fuente: Elaboración propia

Asimismo, en la Figura 16 se grafica la distribución de las variables categóricas: género, profesión, estado civil y ubicación.

Figura 16.Distribución de las variables categóricas



Fuente: Elaboración propia

Para poder aplicar el algoritmo de K-Means es indispensable que todas las variables sean numéricas ya que se necesita calcular la distancia entre los puntos (registros) hacia los centroides de los clústeres. Se usa el método LabelEncoder de la librería ScikitLearn (Pedregosa et al., 2011) para este fin y los resultados se aprecian en la Figura 17.

Las variables tienen diferentes escalas por lo cual es necesario estandarizarlas para evitar el efecto de esta alta variabilidad en la generación de los clústeres (Mohamad & Usman, 2013). En la Figura 18 se muestra el dataset final luego de aplicar el método Scale de la librería ScikitLearn.

Figura 17.Muestra de datos luego de aplicar la codificación a las variables categóricas

edad	genero	ingresos	estado_civil	profesion	ubicacion	deuda_SF	anios_antiguedad	ctd_productos
53	1	26904.037143	0	1	4	38016.483571	29.0	4
65	0	10707.246429	0	1	1	8052.326429	34.0	2
56	1	11754.196071	0	2	5	690366.443929	35.0	1
71	1	8625.447857	0	1	1	48650.166786	45.0	2
58	1	12523.057857	0	1	1	7448.363929	26.0	2

Fuente: Elaboración propia

Figura 18.Muestra de datos luego de aplicar normalización

edad	genero	ingresos	estado_civil	profesion	ubicacion	deuda_SF	anios_antiguedad	ctd_productos
1.195260	0.811801	3.238957	-1.579739	-0.328477	1.671383	-0.130497	2.149097	1.858068
2.278632	-1.231828	0.764734	-1.579739	-0.328477	-0.530184	-0.221164	2.819091	-0.048459
1.466103	0.811801	0.924666	-1.579739	1.460257	2.405239	1.843409	2.953090	-1.001722
2.820318	0.811801	0.446719	-1.579739	-0.328477	-0.530184	-0.098321	4.293078	-0.048459
1.646665	0.811801	1.042118	-1.579739	-0.328477	-0.530184	-0.222991	1.747101	-0.048459

Fuente: Elaboración propia

4.2.3 Selección de atributos

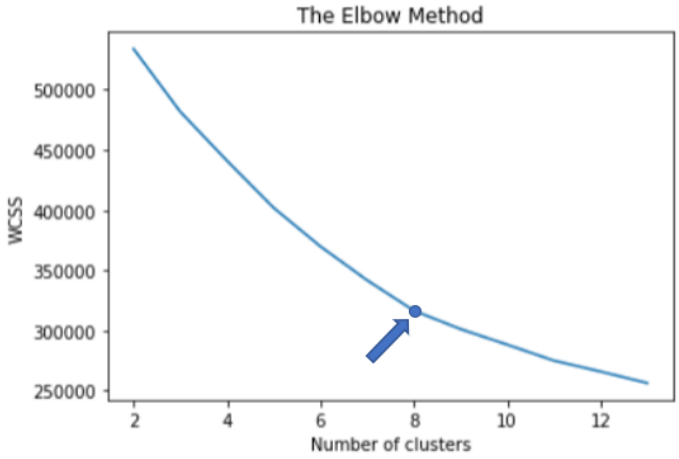
El objetivo de este paso es encontrar el mejor conjunto de variables que permitan generar los clústeres de mejor calidad. Se utiliza el método de información mutua, que forma parte de los métodos de filtrado que se basan en las características estadísticas de las variables. En particular, este método calcula la cantidad de información que se puede obtener de una variable con respecto a la variable objetivo y para efectos de la selección de los mejores atributos se busca que este valor sea el máximo posible (Huijskens, 2017).

Debido a que no se cuenta con la clase, al ser un problema de aprendizaje no supervisado, se decide ejecutar el clustering con K-means usando las 9 variables para poder etiquetar

cada registro con el valor de su clúster correspondiente y calcular la información mutua entre cada una de las variables y la clase para determinar cuáles son las más relevantes.

Para esta primera ejecución se determina el número de clústeres apropiados utilizando el método del codo ("elbow method") que como se puede apreciar en la Figura 19 es de 8.

Figura 19.Número óptimo de clústeres usando el método del codo



Fuente: Elaboración propia

Se utiliza la métrica silhouette para evaluar qué tan compacto es un clúster y qué tan separado está del resto en base al cálculo de distancias. El silhouette para un punto *i* viene dado por la ecuación 9 donde b_i es la distancia promedio del punto *i* al clúster más cercano (distancia inter-clúster) y a_i es la distancia promedio del punto i al resto de puntos del clúster al que pertenece (distancia intra-clúster).

$$s_i = \frac{b_i - a_i}{max(b_i, a_i)} \quad (9)$$

Al promediar los valores de todos los puntos del dataset se obtiene el silhouette total, el cual va de -1 a 1 y se interpreta de la siguiente manera:

- Un valor cercano a 1 significa que el punto pertenece al clúster correcto.
- Un valor cercano a 0 significa que probablemente el punto puede pertenecer a otro clúster.
- Un valor cercano a -1 significa que el punto está en un clúster incorrecto.

Para la configuración descrita, el resultado del silhouette es de 0.3636, lo cual sirve como punto de partida para comparar la calidad del clustering generado luego de la selección de variables.

En la Figura 20 se muestran los resultados de aplicar información mutua sobre las 9 variables con respecto al valor de la clase. Como se visualiza, las 4 variables que más aportan son: edad, el género, ubicación y estado civil.

Figura 20.Cálculo de la información mutua para todas las variables disponibles

	Información mutua
ctd_productos	0.048334
deuda_SF	0.064682
profesion	0.139714
ingresos	0.178878
anios_antiguedad	0.317882
edad	0.348324
genero	0.357957
ubicacion	0.377201
estado_civil	0.438506

Fuente: Elaboración propia

Al seleccionar estas variables y volver a ejecutar el algoritmo de K-Means, esta vez para K = 7, se obtiene un silhouette de 0.654 que supera largamente el resultado obtenido previamente, así como otras pruebas realizadas con diferentes combinaciones de variables que se resumen en la Tabla 9.

Tabla 9.Valores de silhouette para diferentes combinaciones de variables

K	Variables	Silhouette
4	[“genero”, “edad”, “ubicación”, “estado_civil”]	0.5201
4	[“genero”, “edad”, “ubicación”, “estado_civil”, “anios_antiguedad”]	0.4185
4	[“genero”, “edad”, “ubicación”, “estado_civil”, “ingresos”]	0.4774
5	[“genero”, “edad”, “ubicación”, “estado_civil”]	0.6218
5	[“genero”, “edad”, “ubicación”, “estado_civil”, “anios_antiguedad”]	0.4960
5	[“genero”, “edad”, “ubicación”, “estado_civil”, “ingresos”]	0.4940
6	[“genero”,” edad”,” ubicación”,” estado_civil”]	0.6253
6	[“genero”, “edad”, “ubicación”, “estado_civil”, “anios_antiguedad”]	0.5258
6	[“genero”, “edad”, “ubicación”, “estado_civil”, “ingresos”]	0.5338
7	**[“genero”,” edad”,” ubicación”,” estado_civil”]**	**0.6540**
7	[“genero”, “edad”, “ubicación”, “estado_civil”, “anios_antiguedad”]	0.5013
7	[“genero”, “edad”, “ubicación”, “estado_civil”, “ingresos”]	0.5455
8	Todas las variables	0.3636

Fuente: Elaboración propia

4.2.4 Generación de clústeres

Al utilizar la combinación de variables con el mejor silhouette, se obtienen los 7 clústeres cuyos centroides, luego de recuperar las escalas originales, se muestran en la Figura 21.

Figura 21.Valor de los centroides luego de aplicar el clustering

	edad	genero	estado_civil	ubicacion	cantidad_clientes
0	35.77	1.00	2.98	1.09	23147
1	49.93	0.76	0.12	4.33	4232
2	50.01	1.00	0.15	1.06	12018
3	35.24	1.00	2.99	4.36	5924
4	35.21	-0.00	3.00	1.08	17385
5	34.51	-0.00	2.91	4.33	4298
6	48.16	-0.00	0.32	1.09	6489

Fuente: Elaboración propia

A continuación de describen las características de cada uno de los clústeres.

- El clúster 0 agrupa a 23 147 clientes varones con una media cercana a los 36 años, en su gran mayoría solteros y que viven en Lima.
- El clúster 1 contiene 4 232 clientes en su mayoría varones de aproximadamente 50 años, casados que viven en provincias.
- El clúster 2 agrupa a 12 018 clientes varones de 50 años, casados y con residencia en Lima.
- El clúster 3 tiene las mismas características que el clúster 0 con la diferencia de que los 5 924 clientes viven en su mayoría en provincia.
- El clúster 4 también es similar al clúster 0 con la diferencia de que agrupa a 17 385 mujeres.
- El clúster 5 contiene a 4 298 clientes mujeres de aproximadamente 35 años, solteras y que viven en provincia.
- El clúster 6 agrupa a 6 489 clientes mujeres de aproximadamente 48 años, en su mayoría casadas y que viven en Lima.

4.2.5 Entrenamiento del modelo recomendador

Como se comentó al inicio del capítulo, el enfoque del modelo de clustering es predecir el rating de una categoría en base al promedio de calificaciones que recibe dicha categoría por parte de los clientes del clúster correspondiente.

Con este objetivo, del paso anterior se obtiene la relación entre el identificador del cliente y su clúster para poder cruzar con la base de ratings utilizada para el modelo de filtrado colaborativo (Figura 14). Cabe mencionar que antes de realizar el cruce, se divide este último dataset en 80% que será utilizado para el entrenamiento y 20% para la validación del modelo.

Luego, se agrupa por clúster y categoría y se calcula el promedio los ratings para cada grupo con lo cual se obtiene el resultado que se muestra en la Figura 22. Con esto, para cada nuevo cliente, se le asignará el clúster más cercano dependiendo de sus características y se le recomendará las categorías con mejor rating en dicho clúster.

Figura 22.Resultado del modelo basado en clustering

cluster	Categoria	Rating_promedio
0	Alimentacion	4.0
0	Farmacias	3.0
0	Grifos	3.0
0	Operaciones Bancarias	4.0
0	Restaurantes	4.0
0	Servicios de Cable	3.0
0	Servicios de Delivery	3.0
0	Tiendas por Departamento	3.0
0	Transporte	3.0
1	Alimentacion	5.0
1	Farmacias	3.0
1	Grifos	3.0
1	Operaciones Bancarias	4.0
1	Restaurantes	4.0

Fuente: Elaboración propia

4.2.6 Discusión de resultados

Al igual que con el algoritmo basado en filtrado colaborativo, se utilizan las métricas de RMSE y MAE para validar el desempeño del modelo. En este caso, se tuvo que codificar el cálculo de ambas métricas y luego de utilizar el 20% de los clientes como conjunto de prueba se obtuvo un RMSE de 0.8981 y un MAE de 0.6293.

Ambas métricas superan ampliamente los valores del clasificador aleatorio que fueron de 1.4406 para el RMSE y 1.1615 para el MAE e incluso son ligeramente mejores que los resultados obtenidos con el método de filtrado colaborativo basado en ítems que se describen en el apartado 4.1.6. Esto demuestra que el modelo de clustering puede responder adecuadamente a las peticiones de recomendación de clientes nuevos y con ello mitigar el problema del “cold start”.

5. Desarrollo de arquitectura real time

Uno de los objetivos principales de la solución es que el sistema sea capaz de capturar los movimientos de tarjetas en tiempo real y dependiendo de si se trata de un cliente nuevo o no, generar las recomendaciones basadas en el modelo de filtrado colaborativo o en el de clustering.

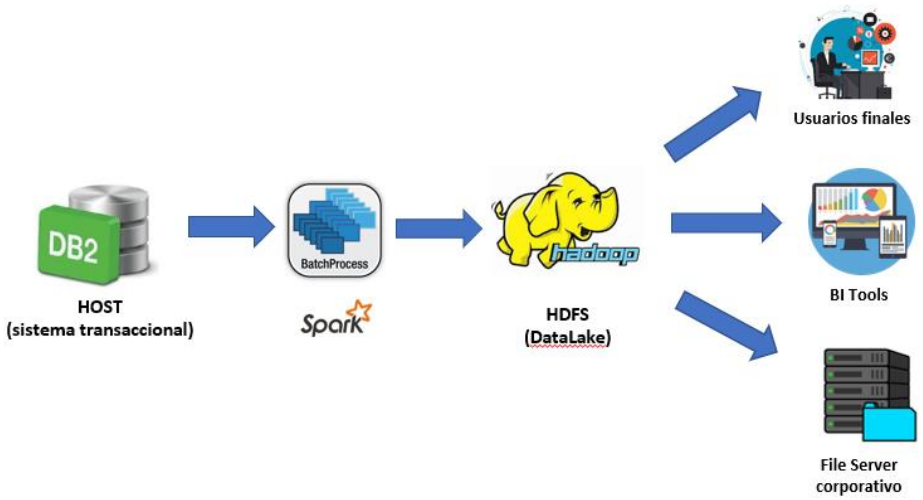
En el presente capítulo, se propone una arquitectura para esta necesidad partiendo de la situación actual de los sistemas que almacenan los datos en la entidad bancaria que se toma como referencia para el caso. Luego se comparan las 2 principales arquitecturas que abordan el tema del procesamiento de datos en tiempo real y se describen las tecnologías a utilizar en cada capa de la arquitectura elegida.

En base a lo anterior, se propone una arquitectura solución y se valida con un ejemplo de cómo funcionaría el sistema recomendador utilizando la arquitectura planteada.

5.1 Contexto de aplicación

Todas las fuentes de datos necesarias para la construcción de los modelos son generadas en los sistemas transaccionales del banco y mediante procesos batch son cargadas a un sistema de ficheros distribuido (HDFS) desde donde está disponible para su consumo por parte de usuarios finales, herramientas de BI y otros sistemas. De manera gráfica se pueden observar los componentes de la arquitectura actual en la Figura 23.

Figura 23.Arquitectura actual de referencia



Fuente: Elaboración propia

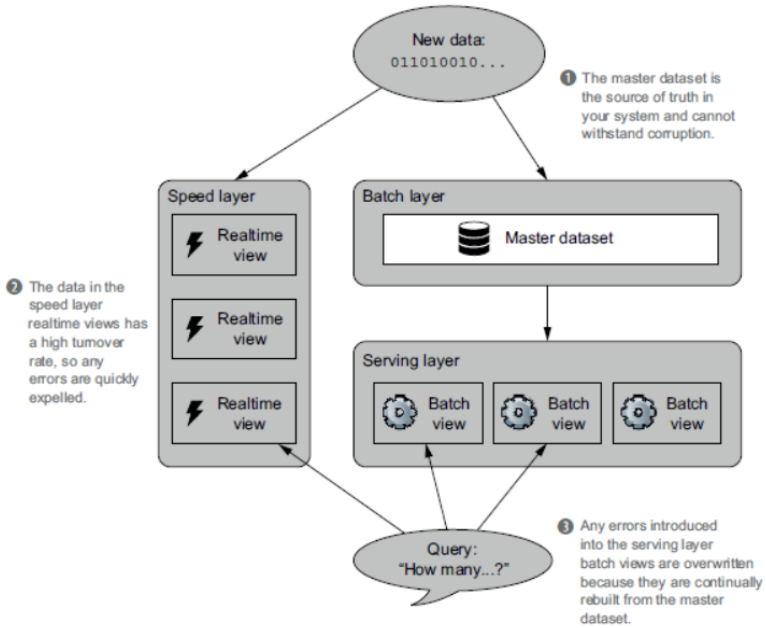
Queda claro que la arquitectura descrita carece de los componentes para soportar la necesidad del procesamiento en tiempo real; sin embargo, se toma como punto de partida ya que se disponen de tecnologías que se pueden reutilizar. Por ejemplo, el sistema de ficheros HDFS se puede usar para el almacenamiento de los modelos entrenados y los resultados de las predicciones, además de Apache Spark como motor de procesamiento de datos distribuido que puede ser utilizado en diferentes etapas del proceso como el aprovisionamiento, preparación de datos, entrenamiento de los modelos y durante la fase de predicción.

5.2 Evaluación de arquitecturas real time

Feick et al. (2018) indican que dos de las principales características de un sistema big data son: tener la capacidad de recibir grandes cantidades de datos, provenientes de diversas fuentes en tiempo real y procesar estos datos para entregar los resultados casi inmediatamente. Para ello describe y compara los 2 patrones arquitectónicos más utilizados en la actualidad: la arquitectura lambda y la arquitectura kappa.

La arquitectura lambda creada por Nathan Marz describe un sistema genérico, escalable, tolerante a fallos y que permite el procesamiento de datos en tiempo real (Marz & Warren, 2015). En la Figura 24 se muestra la interacción entre sus componentes y luego se describen sus principales características.

Figura 24.Diagrama de arquitectura lambda

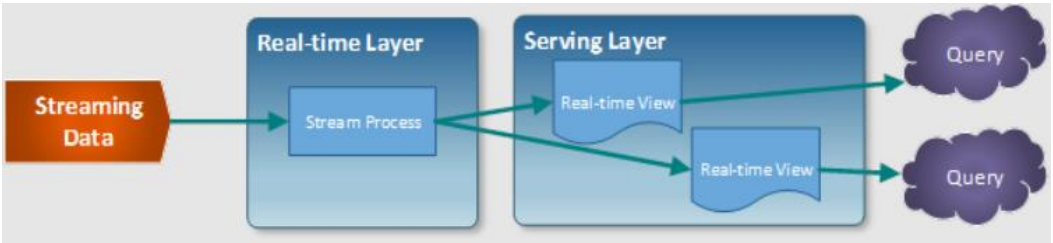


Fuente: (Marz & Warren, 2015)

- La información capturada en tiempo real es enviada tanto a la capa batch como a la capa de streaming.
- La capa batch gestiona los datos en crudo y añade los datos nuevos a los ya existentes para generar un histórico. Luego, a través de un proceso batch, se realizan transformaciones y se generan las vistas que serán usadas por la capa de servicio para mostrar los datos a los usuarios y sistemas finales.
- La capa de streaming o de tiempo real solo tiene en cuenta los datos nuevos y es responsable de indexar estos datos para poder dar respuesta con baja latencia, compensando las demoras que pueda tener la capa batch en su procesamiento.
- La capa de servicio, es la encargada de combinar los resultados de las capas batch y streaming.

Por otro lado, la arquitectura kappa fue propuesta por Jay Kreps como una alternativa más simple a la arquitectura lambda ya que sugiere eliminar la capa batch y quedarse solo con la de tiempo real la cual da servicio a cualquier tipo de consulta del usuario (Kreps, 2014). En la Figura 25 se muestra el diseño de la arquitectura y a continuación Domínguez (2018) describe el funcionamiento de sus componentes.

Figura 25.Diagrama de arquitectura kappa



Fuente: (Domínguez, 2018)

- Todo se considera un “stream” de datos, incluso las operaciones batch las cuales son tratadas como un subconjunto del procesamiento en streaming.
- Los datos son almacenados sin transformar y sirven de input para las vistas en tiempo real que a su vez son la fuente de la capa de servicio.
- Es posible ejecutar un reproceso para variar los resultados obtenidos partiendo de los mismos datos de entrada, para esto es necesario que los datos hayan sido leídos y almacenados en el orden en que se generaron.

- Al existir un único flujo, el esfuerzo de mantener todo el sistema se reduce considerablemente comparado con la arquitectura lambda.

Samizadeh (2018) describe algunas ventajas y desventajas de cada una de las arquitecturas mencionadas las cuales se resumen en la Tabla 10.

Tabla 10.Ventajas y desventajas entre las arquitecturas lambda y kappa

Arquitectura Lambda	**Arquitectura Kappa**
Ventajas: • Cuenta con un almacenamiento distribuido tolerante a fallos para guardar la historia. • Buen balance entre velocidad y confiabilidad. • Arquitectura escalable y que reduce la posibilidad de error incluso si el sistema cae.	Ventajas: • Es mucho menos compleja de mantener e implementar al prescindir de la capa batch. • Útil en sistemas que no necesitan la totalidad los datos sino solo lo que viene en los “streams”. • Solo requiere reprocesar cuando el código cambia.
Desventajas: • El mantenimiento y la sincronización son complejos ya que debe realizarse tanto a la capa batch como a la de streaming. • El reproceso en la capa batch puede no ser la mejor opción en algunos escenarios. • Una arquitectura lambda es difícil de migrar o reorganizar.	Desventajas: • Requiere la gestión de reprocesos o procesos de reconciliación de datos ante errores en el procesamiento. • Puede tener problemas de rendimiento con aplicaciones altamente intensivas como el entrenamiento de un modelo analítico.

Fuente: Elaboración propia

En definitiva, la decisión depende de las características de cada caso de uso y para este en particular se decide utilizar la arquitectura lambda ya que se distingue claramente la necesidad de cada una de sus capas:

- Existe una alta necesidad de procesamiento batch para el aprovisionamiento de las fuentes, preparación de datos y entrenamiento de los modelos.

- Tanto el cálculo de las similitudes entre ítems para el modelo de filtrado colaborativo como la agrupación de los clientes en clústeres pueden realizarse de manera offline lo cual reduce considerablemente el procesamiento que se tiene que hacer en línea.

Claro está que esto requiere una capa batch que entrene y recalibre los modelos periódicamente.

- La capa de tiempo real debe encargarse de capturar el dato del cliente que está haciendo la transacción para decidir con qué modelo realizar la recomendación.
- Se requiere una capa de servicio que pueda tomar el resultado del modelo y presentarlo a los aplicativos de gestión comercial o a los usuarios finales.

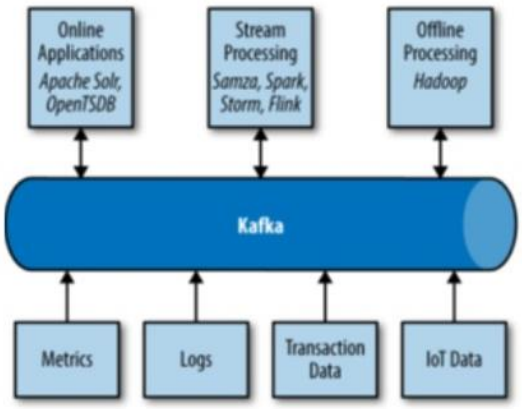
5.3 Elección de tecnologías a utilizar

Para cada una de las capas de una arquitectura lambda existen diversas opciones tecnológicas para su implementación. A continuación, se describen algunas de las principales.

5.3.1 Apache Kafka

Shapira et al. (2021) presentan Kafka como un sistema de mensajería que sigue el patrón publicación/suscripción y que proporciona un canal de datos común (bus de datos) donde varias aplicaciones puedan escribir y leer mensajes tal como se grafica en la Figura 26.

Figura 26.Aplicaciones productoras y consumidoras en un clúster de Kafka



Fuente: (Shapira et al., 2021)

Los mensajes se reciben y escriben como array de bytes en el disco local de las máquinas interconectadas, llamadas brókeres, que forman el clúster de Kafka y son replicados automáticamente lo que aporta robustez frente a fallos, así como un aumento de rendimiento y escalabilidad.

A su vez los mensajes se agrupan en “topics” cuyo símil puede ser una tabla de base de datos o un directorio de un sistema de archivos. El objetivo es que los mensajes con la misma estructura se agrupen y puedan interpretarse de la misma manera. Dentro de cada “topic” se definen particiones que indican cómo se distribuyen físicamente los datos entre los nodos del clúster para efectos de escalabilidad, redundancia y paralelismo al leer o escribir.

5.3.2 Hadoop Distributed File System (HDFS)

Para la capa batch, una de las tecnologías más usadas es HDFS que se define como un sistema de ficheros distribuido destinado a almacenar grandes cantidades de datos utilizando un clúster de ordenadores con hardware convencional (White, 2012).

Sus principales características son: la escalabilidad, ya que si se requiere mayor capacidad basta con añadir más nodos; el almacenamiento de grandes volúmenes de datos, al distribuir los datos entre los diferentes nodos del clúster y el patrón de acceso “write-once, read-many”, que se refiere a que los archivos en HDFS suelen ser creados una vez, rara vez modificados o eliminados y accedidos en numerosas ocasiones.

Adicionalmente, en el ecosistema Hadoop se propone utilizar el paradigma MapReduce para el procesamiento aprovechando que los datos ya se encuentran distribuidos en el clúster (Dean & Ghemawat, 2008). Sin embargo, la complejidad de implementar soluciones solo utilizando las funciones “map” y “reduce”, los problemas de rendimiento asociados a la escritura en disco durante la operación *map* o el movimiento de datos entre nodos durante la operación de *reduce* han hecho que entre en desuso en los últimos años.

5.3.3 Apache Spark

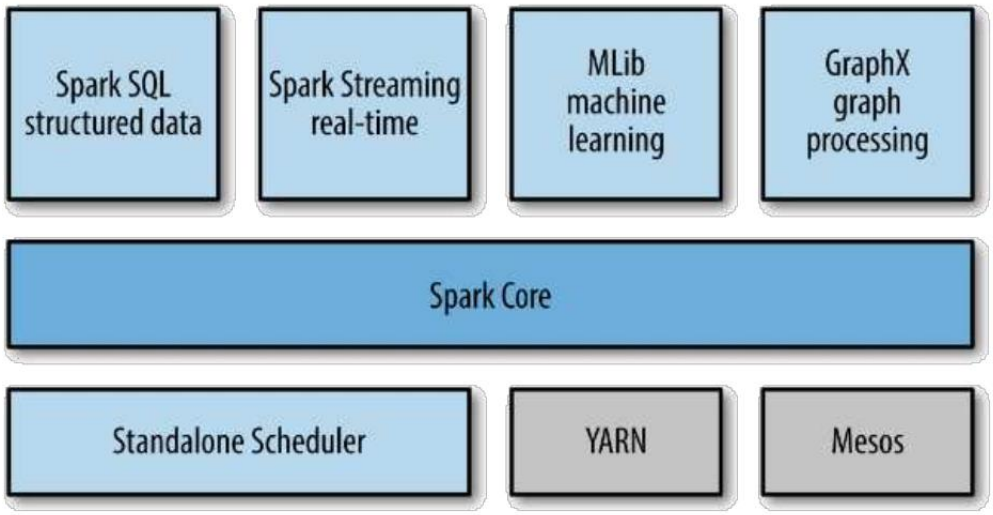
Surge como alternativa a las deficiencias del paradigma MapReduce y Chambers & Zaharia (2018) lo definen como un motor unificado de cálculo en memoria para el procesamiento paralelo y distribuido en un clúster de ordenadores.

Consiste en una API distribuida que opera similar a las consultas SQL tradicionales lo que lo hace fácil de aprender y usar. Se encuentra disponible en 4 lenguajes: Java, Scala, Python y R.

En la Figura 27 se muestra un diagrama de sus componentes. El principal es el Spark Core en el que se encuentran las estructuras fundamentales de Spark llamadas RDD que son las que el motor ejecuta. Los tres módulos inferiores representan los gestores de recursos que se encargan de asignar máquinas, CPUs y memoria a Spark para realizar las tareas.

Con respecto al resto de módulos, Spark SQL contiene una serie de funciones para manejar tablas de datos distribuidas llamadas “dataframes”. Spark Streaming se utiliza para operar datos de manera distribuida en tiempo real. Spark MLlib contiene las implementaciones de algoritmos de machine learning y Spark GraphX es el módulo de procesamiento de grafos.

Figura 27.Componentes de Spark



Fuente: (Chambers & Zaharia, 2018)

La principal ventaja de Spark es que todos los cálculos se realizan en memoria y solo se escriben a disco cuando el usuario lo indica explícitamente o cuando es estrictamente necesario realizar un movimiento de datos entre nodos (shuffle) lo que hace que su rendimiento sea muy superior en tareas iterativas.

5.3.4 Spark Structured Streaming

Como se mencionó líneas arriba, una de las alternativas para la gestión de datos en tiempo real es Spark Structured Streaming. El procesamiento del flujo de datos (stream) consiste en incorporar continuamente nuevos datos para actualizar un resultado. Este recálculo continuo se realiza a través de procesos batch ejecutados en intervalos de tiempo muy cortos que dan la impresión de que se procesan en tiempo real.

Hace uso de la misma API estructurada de Spark y por lo tanto la misma lógica que aplica a los dataframes estáticos funciona para los “streaming dataframes” (dataframes calculados en tiempo real). Los datos de entrada los puede recibir de Kafka, ficheros HDFS o Amazon S3 mientras que la salida se puede escribir en el propio Kafka, en un fichero o en memoria.

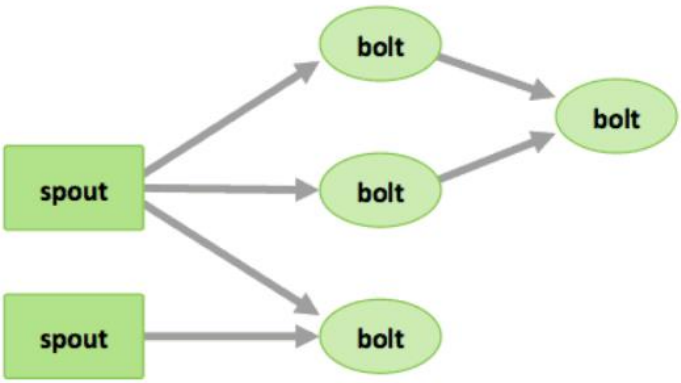
5.3.5 Apache Storm

Storm es un sistema de computación en tiempo real distribuido para el procesamiento de grandes volúmenes de datos en alta velocidad. Es bastante sencillo y los desarrolladores pueden escribir topologías en Storm usando cualquier lenguaje de programación. Entre sus principales características destaca su rapidez, escalabilidad, tolerancia a fallos, fiabilidad y la ya mencionada facilidad de uso (Cloudera, 2022b).

Existen 5 abstracciones que ayudan a comprender cómo Storm procesa los datos:

- Tupla: lista ordenada de elementos.
- Stream: secuencia de tuplas.
- Spouts: fuente de “streams” en un cálculo, por ejemplo, la API de Twitter.
- Bolts: recibe un conjunto de entradas y genera las salidas. Ejecuta funciones como filtros, agregaciones, unión de datos, así como comunicación a bases de datos.
- Topologías: es el cálculo global que se representa de forma visual como un conjunto de spouts y bolts tal como se muestra en la Figura 28.

Figura 28.Ejemplo de topología en Apache Storm



Fuente: (Cloudera, 2022)

5.3.6 Apache Cassandra

Para la capa de servicio una de las alternativas más comunes es la base de datos NoSQL llamada Cassandra. Se caracteriza por ser distribuida, altamente escalable y con alto rendimiento. Es capaz de manejar grandes volúmenes de datos sobre varios servidores garantizando alta disponibilidad, replicación y un número mínimo de fallos (Apache Software Foundation, 2008).

Se trata de una base de datos de tipo clave-valor, que se organiza en tablas y columnas las cuales deben ser diseñadas de acuerdo al tipo de consultas que se van a realizar. Usualmente los datos se encuentran replicados en varias tablas desnormalizadas lo que contribuye a que las lecturas y escrituras sobre Cassandra sean bastante rápidas, sacrificando por otro lado, la capacidad de hacer análisis de datos a través de consultas complejas. Por su rapidez, es utilizada en algunos casos de uso de empresas reconocidas mundialmente como: Facebook, Twitter, Ebay, Netflix entre otras (Carpenter & Hewitt, 2020).

5.3.7 Apache HBase

Es otra de las opciones más utilizadas en la capa de servicio. Cloudera (2022a) describe sus principales características.

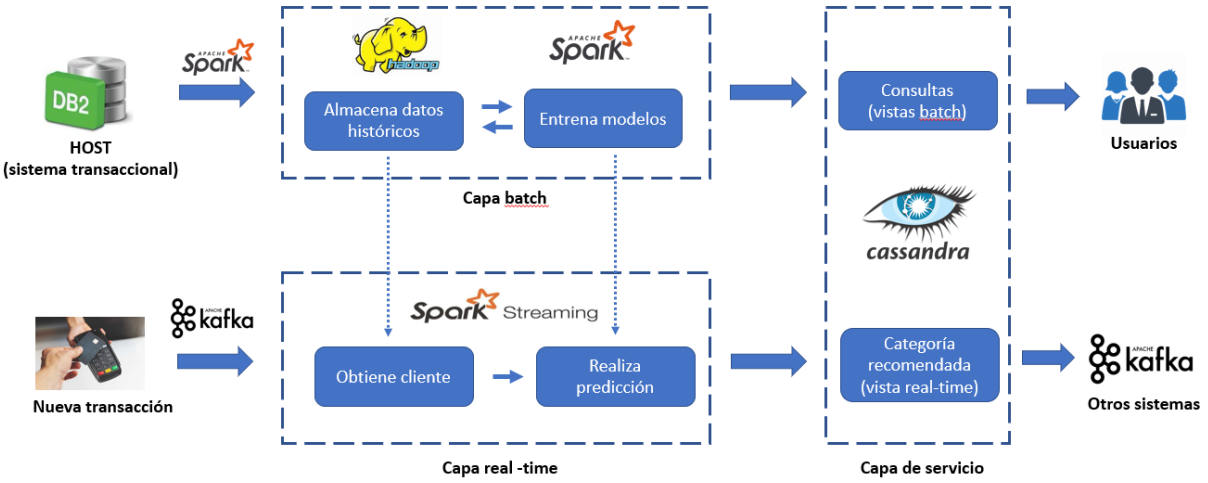
- Responde rápidamente a las consultas realizadas sobre los datos almacenados y se integra con Apache Kafka y Spark Streaming para construir un flujo de trabajo completo sobre la misma plataforma.
- Está diseñado para almacenar grandes cantidades de datos y asegurar la escalabilidad para dar servicio a más usuarios y aplicaciones.
- Permite almacenar diferentes tipos de datos, sean estructurados o no, sin necesidad de realizar un modelado previo. Para el acceso a los datos se integra con tecnologías como Impala o Apache Solr.
- Confiabilidad basada en un proceso de replicación automática que protege los datos ante pérdidas y una gestión de recuperación ante desastres que permite continuar con el negocio a pesar de experimentar alguna incidencia.

5.4 Arquitectura propuesta

Una vez revisadas las opciones tecnológicas para implementar la arquitectura lambda y tomando en cuenta la situación actual de los sistemas que gestionan los datos en la entidad bancaria, se propone la arquitectura mostrada en la Figura 29.

A continuación, se describen las diferentes capas y las tecnologías escogidas para cada una:

Figura 29.Arquitectura propuesta



Fuente: Elaboración propia

Con respecto a las fuentes de datos, se tienen las que se generan en los sistemas transaccionales del banco (maestros de cliente, relación cliente-contrato, maestro de códigos MCC, entre otras) y que ya son llevadas diariamente al datalake corporativo a través de cargas batch. En esta parte se reutilizan los procesos Spark que extraen los datos de las fuentes y las cargan al datalake.

Por otro lado, la fuente principal sobre la cual se construyen los modelos son las transacciones que hacen los clientes con sus tarjetas y que son capturadas en tiempo real utilizando una cola de mensajes de Kafka. Se elige esta tecnología por ser escalable, robusta y de fácil integración con otras tecnologías propuestas en esta arquitectura como Spark.

En la capa batch, como se mencionó, la entidad bancaria ya cuenta con un datalake construido sobre un sistema de ficheros distribuido de Hadoop (HDFS) que almacena toda la historia requerida para las fuentes de los modelos de filtrado colaborativo y clustering. Cabe mencionar que ambos modelos serán entrenados diariamente de manera batch y recibirán las peticiones de inferencia con la versión que haya sido calculada para ese día específico. Toda la preparación de datos y el procesamiento de los modelos se hará en el clúster de Apache Spark que se tiene disponible.

En la capa de tiempo real, se reciben las transacciones provenientes de Kafka y se recupera el código del cliente utilizando los datos que se encuentran almacenados en el datalake. Una vez identificado el cliente y dependiendo de si este es nuevo o no, se realiza la predicción del rating para las categorías de consumo a través de los modelos de filtrado colaborativo o clustering previamente entrenados en la capa batch. Se elige Spark

Streaming sobre Apache Storm debido a que ya se cuenta con un clúster de Spark desplegado en la organización y se puede utilizar la misma lógica de la API estructurada tanto para dataframes estáticos (comunes en las cargas batch) como para los “streaming dataframes” necesarios para implementar la lógica descrita.

Uno de los principales objetivos del trabajo es presentar la recomendación de las mejores categorías que el cliente aún no ha consumido para poder iniciar una campaña personalizada. Esto en una arquitectura lambda se le conoce como una vista en tiempo real y es gestionada por una capa de servicio que a su vez también puede manejar vistas batch, por ejemplo, un agregado de las categorías más recomendadas en el último mes.

Para poder atender este tipo de peticiones, que pueden venir de algún sistema del banco o de un usuario específico, se propone Apache Cassandra por su baja latencia para resolver las consultas, escalabilidad para manejar grandes volúmenes de datos, alta disponibilidad ante fallos, posee un lenguaje similar a SQL para las consultas de los usuarios y se integra fácilmente con Spark y Kafka, tecnologías presentes en la propuesta.

5.5 Validación de la arquitectura

Para validar la factibilidad de implementar la arquitectura propuesta, en este apartado se detalla una demostración del funcionamiento del sistema recomendador integrando las tecnologías descritas en el punto anterior.

En la Figura 37 se muestra un fragmento del código utilizado en la generación del algoritmo para calcular el rating implícito como fuente del modelo de filtrado colaborativo. Tanto las fuentes originales como las generadas durante la preparación de datos se leen desde HDFS y se procesan de manera distribuida en el clúster de Spark. Para interactuar con Spark se utiliza el lenguaje de programación Python por su facilidad de uso y porque cuenta con una serie de librerías para desarrollar modelos de machine learning.

Figura 30.Uso de Apache Spark en la preparación de datos

```
dfRankFreq = dfFiltrado.groupBy("customer_id","Categoria").\
             agg(F.count("*").alias("conteo"),F.min("diferencia_dias").alias("dias_ult_compra"),
                 F.sum("transaction_amount").alias("total_monto")).\
             withColumn("rank_freq", F.rank().over(window)).\
             withColumn("rank_monto", F.rank().over(window2)).\
             withColumn("Rating_x_Frecuencia",F.when(F.col("rank_freq")==1,5).\
                        when(F.col("rank_freq").isin(2,3),4).\
                        when(F.col("rank_freq").isin(4,5),3).\
                        when(F.col("rank_freq").isin(6,7),2).\
                        when(F.col("rank_freq").isin(8,9),1)).\
             withColumn("Rating_x_monto",F.when(F.col("rank_monto")==1,5).\
                        when(F.col("rank_monto").isin(2,3),4).\
                        when(F.col("rank_monto").isin(4,5),3).\
                        when(F.col("rank_monto").isin(6,7),2).\
                        when(F.col("rank_monto").isin(8,9),1)).\
             withColumn("Rating_x_frescura",F.when(F.col("dias_ult_compra") <= 60,5).\
                        when(F.col("dias_ult_compra") <= 120,4).\
                        when(F.col("dias_ult_compra") <= 180,3).\
                        when(F.col("dias_ult_compra") <= 360,2).\
                        when(F.col("dias_ult_compra") > 360,1)).\
             withColumn("Rating_ponderado",F.round(F.col("Rating_x_Frecuencia")*Frecuencia +
                                                   F.col("Rating_x_monto")*Monto +
                                                   F.col("Rating_x_frescura")*Frescura))
```

customer_id	Categoria	Rating_x_Frecuencia	Rating_x_monto	Rating_x_frescura	Rating_ponderado
24673321	Tiendas por Depar...	2	5	5	4.0
24673321	Operaciones Banca...	3	4	5	4.0
24673321	Alimentacion	4	4	5	4.0
24673321	Transporte	5	3	5	4.0
24673321	Restaurantes	4	3	5	4.0
24673321	Servicios de Cable	3	2	5	3.0
24673321	Farmacias	2	2	3	2.0
24673321	Servicios de Deli...	1	1	5	2.0
24673321	Grifos	1	1	2	1.0

Fuente: Elaboración propia

Como se indicó en el apartado anterior, los modelos son entrenados a través de procesos batch y son invocados por las solicitudes de predicción que llegan en tiempo real. Para el caso del modelo de filtrado colaborativo se utiliza una librería especializada en sistemas recomendadores llamada Surprise que recibe un dataset con el código del cliente, el producto y el rating asociado. Por el lado del modelo de clustering se utiliza la librería KMeans de Scikit-learn que requiere el dataset con los atributos seleccionados y el número de clústeres a generar. En la Figura 31 se muestran ambas implementaciones hechas en Python y ejecutadas sobre el clúster de Spark.

Figura 31.Entrenamiento del modelo de clustering (izq.) y modelo de filtrado colaborativo (der.)

```
##Construcción de Clusters
n = 6
import warnings
warnings.filterwarnings('ignore')
clu = KMeans(n_clusters = n,
      init = "k-means++", random_state = 10)
kmeans = clu.fit(data_sd)
kmeans
#data_sd['cluster'] = kmeans.labels_
#data_sd.head()
```

```
KMeans(n_clusters=6, random_state=10)
```

```
# Entrenamiento de modelo con el algoritmo KNNWithMeans
trainset = data.build_full_trainset()
sim_options = {'name': 'cosine',
               'user_based': False  # similitud entre ítems
               }
algo = KNNWithMeans(sim_options=sim_options)
algo.fit(trainset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x7f0b0deb0470>
```

Fuente: Elaboración propia

En la capa de tiempo real se utiliza Apache Kafka para gestionar el flujo de datos generado por las transacciones con tarjeta. Se crea un productor de Kafka que escribe los mensajes

en el “topic” llamado “transacciones” para que pueda ser consumido desde Spark Streaming y así recuperar el código de cliente utilizando un dataframe en tiempo real. La simulación de este flujo se muestra en las Figuras 32 y 33.

Figura 32.Generación de mensajes desde un productor de Kafka y consumo desde un streaming dataframe de Spark

```
>/usr/lib/kafka/bin/kafka-console-producer.sh --broker-list clusterrealtime-w-0:9092 --topic transacciones
>1235,465832020345,65.98,5812,Restaurante El Hornero,2022-05-26
>
```

```
#Se crea el dataframe que lee de Kafka
streamingDF = spark.readStream.format("kafka")\
                    .option("kafka.bootstrap.servers",
                            "clusterrealtime-w-0:9092,clusterrealtime-w-1:9092")\
                    .option("subscribe", "transacciones")\
                    .load()
```

Fuente: Elaboración propia

Figura 33.Configuración de escritura del streaming dataframe (izq.) y obtención del código de cliente en tiempo real (der.)

```
customerIDStreaming = getCustomer(parsedDF)

consoleOutput = customerIDStreaming\
                    .writeStream\
                    .queryName("obtiene_cliente")\
                    .outputMode("update")\
                    .format("memory")\
                    .start()
```

```
cliente = spark.sql("select * from obtiene_cliente")
cliente.show()
```

```
+-----------+
|customer_id|
+-----------+
|   20707652|
+-----------+
```

Fuente: Elaboración propia

En este punto, el sistema debe decidir si realizar la predicción utilizando el método de filtrado colaborativo o el de clustering. Si el cliente existe en la base de datos con la que se entrenó el modelo de filtrado colaborativo, se realizará la predicción de todas aquellas categorías que aún no ha consumido. Como se observa en la Figura 34, los resultados de la predicción se muestran ordenados de mayor a menor, es decir, las primeras categorías son las que tienen mejor rating y por lo tanto serían las más adecuadas para gestionar una oferta personalizada.

Figura 34.Resultados de la predicción de cada categoría que el cliente aún no ha consumido

usuario	categoria	valorReal	valorPredicho
20707652	Operaciones Bancarias	null	4.233119
20707652	Transporte	null	3.4516342
20707652	Grifos	null	3.3979208
20707652	Servicios de Delivery	null	3.353865
20707652	Servicios de Cable	null	3.0987039

Fuente: Elaboración propia

El otro escenario se da cuando un cliente no ha consumido ninguna de las 9 categorías consideradas en el estudio y se realiza la recomendación a través del modelo de clustering. En la Figura 35 se muestra la captura de una transacción que corresponde a un cliente nuevo, en este caso, se recuperan los atributos edad, género, ubicación y estado civil para poder clasificarlo en uno de los clústeres previamente generados. Una vez identificado el clúster, el sistema calcula los ratings para todas las categorías y las presenta en el mismo formato como se puede visualizar en la Figura 36.

Figura 35.Captura de transacción de un cliente nuevo y recuperación de atributos requeridos para el modelo de clustering

```
>5431,493050392201,154.38,5514,Grifo Primax,2022-05-26
```

```
cliente = spark.sql("select * from obtiene_cliente")
cliente.show()
```

customer_id
12345678

```
getCustomerFields(cliente).show()
```

customer_id	edad	genero	ubicacion	estado_civil
12345678	53	M	OTRO	CASADO

Fuente: Elaboración propia

Figura 36.Resultados de la predicción para cada categoría usando el modelo

usuario	categoria	valorReal	valorPredicho
12345678	Alimentacion	null	5.0
12345678	Operaciones Bancarias	null	4.0
12345678	Restaurantes	null	4.0
12345678	Grifos	null	3.0
12345678	Servicios de Delivery	null	3.0
12345678	Transporte	null	3.0
12345678	Tiendas por Departamento	null	3.0
12345678	Farmacias	null	3.0
12345678	Servicios de Cable	null	3.0

Fuente: Elaboración propia

Como último paso, se implementa la capa de servicio en una base de datos Cassandra donde se persisten las vistas en tiempo real, es decir las predicciones (Figuras 34 y 36) y también algunas vistas batch que se requieran a demanda. Estos datos pueden ser consumidos por otras aplicaciones en tiempo real, para lo cual se propone el uso de Kafka, o consultados directamente por usuarios a través del lenguaje CQL propio de Cassandra. En la Figura 37 se muestra el uso del conector Spark Cassandra usado para cargar los datos a Cassandra desde Spark y una vista de la tabla resultante.

Figura 37.Uso del conector Spark Cassandra y vista final de la tabla con las predicciones en Cassandra

```
## Logica para escribir en Cassandra
dfPrediccion.write
  .cassandraFormat("prediccion_cf", "modelo")
  .mode("append")
  .save()
```

```
cqlsh:modelos> select * from prediccion_cf ;

 usuario  | categoria             | valorpredicho | valorreal
----------+-----------------------+---------------+-----------
 20707652 |                Grifos |       3.39792 |      null
 20707652 | Operaciones Bancarias |       4.23312 |      null
 20707652 |    Servicios de Cable |        3.0987 |      null
 20707652 | Servicios de Delivery |       3.35386 |      null
 20707652 |            Transporte |       3.45163 |      null
```

Fuente: Elaboración propia

6. Conclusiones y trabajo futuro

6.1 Conclusiones

En el capítulo 3 se planteó el objetivo principal de diseñar un sistema recomendador que trabaje en tiempo real y que permita a los equipos comerciales gestionar una oferta personalizada para un cliente en base a la transaccionalidad de sus tarjetas de crédito o débito y a otras variables disponibles. A lo largo del desarrollo de la contribución se cumple este objetivo ya que se presenta un sistema recomendador basado en modelos de filtrado colaborativo y clustering sobre una arquitectura que permite realizar predicciones en tiempo real y con ello aprovechar el momento en que el cliente interactúa con el banco para ofrecer una recomendación.

Los objetivos específicos se derivan de los principales problemas que enfrenta un sistema recomendador, en especial en el sector bancario, que son la carencia de ratings debido a que los clientes no suelen calificar los productos que consumen lo que genera el llamado problema del “data sparsity” y la imposibilidad de dar una recomendación a un cliente nuevo del que aún no se conocen sus preferencias, lo que se conoce como “cold start”.

El trabajo tomó como datos las transacciones de tarjetas de más de 70 mil clientes y se utilizó el enfoque de Kaya et al. (2021) para clasificar los movimientos en 9 categorías de consumo sobre las cuales se hicieron las predicciones. El problema de la falta de ratings se abordó con la propuesta de un algoritmo que calcula el rating implícito para cada categoría utilizando las variables del modelo RFM: frescura, frecuencia y monto de transacciones (Sharifihosseini, 2019). Con esto se alimentó un modelo de filtrado colaborativo basado en la similitud de ítems ya que este puede ser calculado de antemano reduciendo la necesidad de procesamiento al momento de hacer la predicción en línea.

Con respecto al problema de generar recomendaciones a clientes nuevos, se diseñó un modelo basado en clustering y se obtuvo que los atributos más significativos para clasificar a los clientes son 4: edad, género, estado civil y ubicación. Con estos se generaron 7 clústeres que permitieron predecir el rating para cada categoría utilizando el promedio de las calificaciones de los clientes de cada clúster. Del mismo modo que el modelo anterior, estos clústeres son generados previamente lo que reduce los tiempos de respuesta al momento de hacer la predicción en tiempo real.

Para medir la validez del modelo se utilizó la escala de ratings de 1 a 5 que es la más utilizada en el ámbito de sistemas recomendadores y se calculó el error en la predicción de

cada modelo utilizando las métricas MAE y RMSE. Los resultados demostraron que la propuesta de usar las 3 variables del modelo RFM mejora en cerca del 12% al enfoque de Oyebode & Orji (2020) que solo considera la frescura en el cálculo del rating implícito. Además, se validó que es factible utilizar los modelos de filtrado colaborativo y clustering propuestos para la recomendación de categorías de consumo y posterior gestión de ofertas comerciales dentro del ámbito bancario ya que se obtuvo una mejora de más del 30% sobre un predictor aleatorio y el promedio del error osciló entre 0.9 y 1 lo cual es aceptable considerando la escala de ratings mencionada.

Otro de los objetivos específicos de la propuesta fue que el sistema recomendador pueda capturar las transacciones, procesarlas y realizar la predicción en tiempo real. Para ello se evaluó la situación actual de los sistemas que gestionan los datos de la entidad bancaria con el objetivo de reutilizar algunas tecnologías.

Luego se realizó una comparativa entre los 2 principales enfoques que abordan el tema del procesamiento de datos en tiempo real: arquitectura lambda y arquitectura kappa. Se eligió la primera principalmente por la necesidad de contar con una capa batch, la cual no existe en la arquitectura kappa, para el aprovisionamiento de datos y el entrenamiento de los modelos.

Se revisaron las principales tecnologías disponibles para cada capa y en la propuesta de la arquitectura se eligió a HDFS con Spark para el almacenamiento y procesamiento en la capa batch, Kafka con Spark Streaming para la captura de las transacciones y el procesamiento en tiempo real y Apache Cassandra como persistencia de la capa de servicio para atender tanto las peticiones de otras aplicaciones como la de usuarios ya sea en modo batch o en tiempo real.

Para validar el diseño de la arquitectura, se mostró un ejemplo de cómo funcionaría el sistema recomendador al recibir una transacción de un cliente nuevo y de uno ya existente comprobándose el flujo utilizado para realizar la predicción en cada caso y la integración de las tecnologías elegidas.

6.2 Líneas de trabajo futuro

Para futuros trabajos se propone agregar en el modelo de filtrado colaborativo algunas variables adicionales como la ubicación de los clientes o información acerca de sus hábitos para que el sistema recomendador pueda generar predicciones más acertadas. Asimismo, implementar el sistema propuesto permitiría obtener feedback de los usuarios lo cual puede ser tomado como una métrica para medir la utilidad del recomendador y serviría para ajustar el modelo.

Con respecto a los modelos utilizados, puede ser interesante explorar los algoritmos de filtrado colaborativo basados en modelos de factorización matricial siendo los más conocidos SVD, PMF y NMF para la reducción de dimensionalidad de la matriz cliente-producto y ALS para la optimización de la función objetivo (Yang et al., 2016). Otra opción en esta línea es el uso de algoritmos basados en redes neuronales como alternativa al filtrado colaborativo.

Probar otras ponderaciones, otras variables o incluso usar modelos de machine learning para el cálculo de los ratings implícitos pueden ser alternativas para atacar el problema del "data sparsity". Por otro lado, para el tema del clustering se pueden validar otros métodos de filtrado para la selección de atributos como el coeficiente de correlación, el score de Fisher o el umbral de varianza, métodos "wrapper" y métodos embebidos como la regresión Lasso o arboles de decisión (Gupta, A., 2020).

7. Bibliografía

References

Apache Software Foundation. (2006). *Hadoop.* https://hadoop.apache.org/

Apache Software Foundation. (2008a). *Cassandra.* https://cassandra.apache.org/_/index.html

Apache Software Foundation. (2008b). *CouchDB.* https://couchdb.apache.org/#more

Apache Software Foundation. (2014a). *Kafka.* https://kafka.apache.org/

Apache Software Foundation. (2014b). *Storm.* https://storm.apache.org/

Apache Software Foundation. (2015). *Ignite.* https://ignite.apache.org/

Barrenetxea, G. (2022). *Los sistemas de recomendación (II): Modelos para enriquecer el customer journey.* https://solvenup.com/modelos-para-enriquecer-el-customer-journey/

Beheshti, A., Yakhchi, S., Mousaeirad, S., Ghafari, S. M., Goluguri, S. R., y Edrisi, M. A. (2020). Towards cognitive recommender systems. *Algorithms, 13*(8), 176.

Chandramouli, B., Levandoski, J. J., Eldawy, A., & Mokbel, M. F. (2011). (2011). Streamrec: A real-time recommender system. Paper presented at the *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data,* 1243-1246.

Choi, K., Yoo, D., Kim, G., y Suh, Y. (2012). A hybrid online-product recommendation system: Combining implicit rating-based collaborative filtering and sequential pattern analysis. *Electronic Commerce Research and Applications, 11*(4), 309-317.

Darnell, B., & Taylor, B. (2009). *Tornado web server.* https://www.tornadoweb.org/en/stable/

Davenport, T. H., Mule, L. D., y Lucker, J. (2011). Know what your customers want before they do. *Harvard Business Review, 89*(12), 84-92.

Gallego Vico, D., Huecas Fernández Toribio, G., y Salvachúa Rodríguez, J. (2012). Generating context-aware recommendations using banking data in a mobile recommender system.

Ghosal, A., Nandy, A., Das, A. K., Goswami, S., y Panday, M. (2020). A short review on different clustering techniques and their applications. *Emerging Technology in Modelling and Graphics,* , 69-83.

Goldstein, D., & Lee, Y. (2005). The rise of right-time marketing. *Journal of Database Marketing & Customer Strategy Management, 12*(3), 212-225.

GroupLens. (2019). *MovieLens dataset.* https://grouplens.org/datasets/movielens/

Gupta, J., & Gadge, J. (2014). (2014). A framework for a recommendation system based on collaborative filtering and demographics. Paper presented at the *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA),* 300-304.

Hernández-Nieves, E., Hernández, G., Gil-González, A., Rodríguez-González, S., y Corchado, J. M. (2020). Fog computing architecture for personalized recommendation of banking products. *Expert Systems with Applications, 140*, 112900.

Huang, Y., Cui, B., Zhang, W., Jiang, J., & Xu, Y. (2015). (2015). Tencentrec: Real-time stream recommendation in practice. Paper presented at the *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data,* 227-238.

Hug, N. (2019). *Surprise A python scikit for recommender systems.* http://surpriselib.com/

IBM. (2022). *Sistemas de recomendación - filtrado colaborativo.* https://es.coursera.org/lecture/aprendizaje-automatico-con-python/filtrado-colaborativo-4y9l1

Jason Lu, Navni Agarwal, & Snigdha Bhardwaj. (2022). *The next best action for banks - chapter 3: Formulating recommendations.* https://www.wwt.com/article/the-next-best-action-for-banks-chapter-3-formulating-recommendations

Kaya, T. S., Gezer, M., & Gülseçen, S. (2021). (2021). Application of recommender system for spending habits based campaign management. Paper presented at the *Multidisciplinary Digital Publishing Institute Proceedings, , 74*(1) 7.

Kim, J., Yoo, J., Lim, H., Qiu, H., Kozareva, Z., & Galstyan, A. (2013). (2013). Sentiment prediction using collaborative filtering. Paper presented at the *Proceedings of the International AAAI Conference on Web and Social Media, , 7*(1) 685-688.

Klioutchnikov, I. K., Klioutchnikov, O. I., y Molchanova, O. A. (2020). Financial intermediary recommender systems. *IBIMA Business Review, 2020*, 182034.

Lops, P., Gemmis, M. d., y Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook,* , 73-105.

Megargel, A., Shankararaman, V., & Reddy, S. K. (2018). Real-time inbound marketing: A use case for digital banking. *Handbook of blockchain, digital finance, and inclusion, volume 1* (pp. 311-328). Elsevier.

Mokarrama, M. J., Khatun, S., y Arefin, M. S. (2020). A content-based recommender system for choosing universities. *Turkish Journal of Electrical Engineering & Computer Sciences, 28*(4), 2128-2142.

MongoDB Inc. (2009). *MongoDB.* https://www.mongodb.com/

Numnonda, T. (2018). A real-time recommendation engine using lambda architecture. *Artificial Life and Robotics, 23*(2), 249-254.

Oyebode, O., & Orji, R. (2020). A hybrid recommender system for product sales in a banking environment. *Journal of Banking and Financial Technology, 4*(1), 15-25.

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review, 13*(5), 393-408.

Popat, S. K., & Emmanuel, M. (2014). Review and comparative study of clustering techniques. *International Journal of Computer Science and Information Technologies, 5*(1), 805-812.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). (2001). Item-based collaborative filtering recommendation algorithms. Paper presented at the *Proceedings of the 10th International Conference on World Wide Web,* 285-295.

Shahapure, K. R., & Nicholas, C. (2020). (2020). Cluster quality analysis using silhouette score. Paper presented at the *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA),* 747-748.

Shankararaman, V., & Megargel, A. (2013). Enterprise integration: Architectural approaches. *Service-driven approaches to architecture and enterprise integration* (pp. 67-84). IGI Global.

Sharaf, M., Hemdan, E. E., El-Sayed, A., y El-Bahnasawy, N. A. (2022). A survey on recommendation systems for financial services. *Multimedia Tools and Applications,* , 1-21.

Sharifihosseini, A. (2019). (2019). A case study for presenting bank recommender systems based on bon card transaction data. Paper presented at the *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE),* 72-77.

Singhal, R., Shroff, G., Kumar, M., Choudhury, S. R., Kadarkar, S., Virk, R., Verma, S., & Tewari, V. (2019). (2019). Fast online'next best offers' using deep learning. Paper presented at the *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data,* 217-223.

Sunny, B. K., Janardhanan, P. S., Francis, A. B., & Murali, R. (2017). (2017). Implementation of a self-adaptive real time recommendation system using spark machine learning libraries. Paper presented at the *2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES),* 1-7.

Van Rossum, G. (1995). Python reference manual. *Department of Computer Science [CS],* (R 9525)

VISA Commercial Solutions. (2007). *Merchant category codes for IRS form 1099-MISC reporting.* https://web.archive.org/web/20070710202209/http://usa.visa.com/download/corporate/resources/mcc_booklet.pdf

Wei, J., Lin, S., y Wu, H. (2010). A review of the application of RFM model. *African Journal of Business Management, 4*(19), 4199-4206.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., y Franklin, M. J. (2016). Apache spark: A unified engine for big data processing. *Communications of the ACM, 59*(11), 56-65.

Zibriczky, D.Recommender systems meet finance: A literature review. Paper presented at the *Proc. 2nd Int. Workshop Personalization Recommender Syst,* 1-10.