Ejercicio de evaluación: Spark Streaming

Alejandro Mira Abad

[1] Universidad Internacional Menéndez Pelayo, España
100007767@alumnos.uimp.es

Abstract. El presente documento pretende documentar el trabajo realizado para el entregable Streaming (Spark Streaming) de la asignatura Big Data: Herramientas para el procesamiento de datos masivos del máster AEPIA-UIMP. El trabajo a costado del diseño e implementación de un sistema capaz de detectar datos anómalos en un flujo de datos continuo. Se parte de un esqueleto con las funcionalidades básicas para poder llevar a cabo el desarrollo del trabajo.

Keywords: Big Data, Spark Streaming, Kafka.

1 Introducción

El trabajo se ha desarrollado utilizando tecnologías dedicadas al análisis y procesamiento de flujos de datos, estas son **Apache Spark Streaming** y **Apache Kafka**. Estas herramientas han sido fundamentales para la implementación y gestión eficiente de flujos de datos en tiempo real. Además, es importante destacar que el proyecto ha sido desarrollado en el lenguaje de programación Scala, el cual ha brindado un alto nivel de concisión y potencia para la manipulación de datos a gran escala. El trabajo realizado se puede encontrar adjunto a la presente memoria o en el repositorio de GitHub creado para el mismo[1].

El supuesto caso planteado trata de detectar facturas anómalas, con el objetivo de poder detectar posibles fraudes u oportunidades de ofrecer ofertas a clientes especiales. En base a un feed de compras, el sistema debe procesar el flujo de datos entrante y procesarlos. Se han pre-entrenado dos modelos de clustering, KMeans y Bisection-KMeans, para detectar las deseadas facturas anómalas. Paralelamente el sistema identifica y separa las facturas que presentan algún problema (el id del cliente está vacío, falta la fecha, etc), a su vez el sistema notifica el número de facturas canceladas (se identifican porque el campo “InvoiceNo” empieza por “C”) en los últimos 8 minutos (actualizando cada minuto).

El pipeline descrito se puede describir mediante la siguiente figura:

[1] Repositorio de GitHub: https://github.com/almiab1/MUIIABigDataStreaming

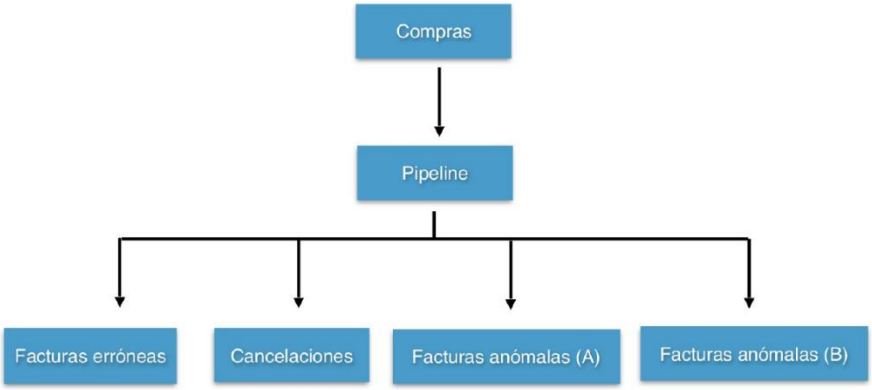

Fig. 1. Pipeline propuesto para el ejercicio.

En la figura (Fig.1) se muestra como el feed de "compras" envía los registros al pipeline el cual procesará y analizará los datos enviando el resultado del análisis a los distintos topic, facturas erróneas, cancelaciones, facturas anómalas (A) y facturas anómalas (B). En el tópico de "facturas erróneas" se publican aquellas facturas con algún dato erróneo como se ha descrito anteriormente. En el caso del tópico "cancelaciones" se publican el número de facturas canceladas en los últimos 8 minutos. Por último, en los tópicos "facturas anómalas A" y "facturas anómalas B" se publican los resultados de haber aplicado los modelos pre entrenados a los datos recibidos.

2 Modelos de clustering y detección de anomalías

Previo a la implementación del pipeline donde se aplica la lógica diseñada para el caso descrito es necesario construir los modelos de clustering que servirán para una posterior detección de anomalías. En este caso se ha actuado en tres aspectos fundamentales de la definición e implementación de los modelos, las fases de selección de atributos y filtrado de los mismos. A su vez se ha implementado el método de selección de parámetros ***elbow*** con error intracluster. El flujo que siguen ambos modelos planteados es el mismo, inicialmente se cargan los datos, éstos pasan por un proceso de selección de características, posteriormente se filtran los datos resultantes siendo estos lo que recibirá el modelo final.

En el proyecto se encuentran en el paquete *es.dmr.uimp.clustering* los scripts correspondientes a la sección de clustering. En específico se encuentran 3 scripts, uno donde se encuentran los métodos principales que servirán para la fase de entrenamiento y otros dos que corresponden al flujo de definición y entrenamientos de los modelos propuestos (KMeans y BisectionKMeans). Con el fin de unificar el proceso de entrenamiento, paralelamente se ha creado un script a través del cual ejecutar los procesos de entrenamiento de clustering.

2.1 Selección de características

Dado que los datos recibidos corresponden a compras realizadas, es necesario agrupar estas compras según su factura correspondiente. Durante este proceso de agrupación, se llevan a cabo varias acciones, como el cálculo del precio promedio, máximo y mínimo. Una vez que las compras se han agrupado, se obtiene un conjunto de facturas definidas por los siguientes atributos:

- **InvoiceNo**: Identificador de la factura.
- **AvgUnitPrice**: Precio promedio de las compras realizadas en la factura.
- **MinUnitPrice**: Precio mínimo de las compras realizadas en la factura.
- **MaxUnitPrice**: Precio máximo de las compras realizadas en la factura.
- **NumberItems**: Cantidad de elementos comprados en la factura.
- **Time**: Hora del día en que se realizó la factura.

2.2 Filtrado de facturas erróneas

En este caso se quiere limpiar el dataset entreat para mantener sólo aquellas facturas que estén correctas. Es por ello que se realiza un filtrado del mismo, eliminando aquellas facturas cuyos atributos contengan algún error. Se ha considerado como primera norma, que un dato es erróneo cuando es nulo. Del mismo modo, para los atributos numéricos como "*AvgUnitPrice*", "*MinUnitPrice*", "*MaxUnitPrice*", "*NumberItems*" o "*Time*" se consideran erróneos cuando su valor es menor que 0. Finalmente, dado que pueden haber facturas canceladas, estas también se consideran como no deseadas en el conjunto final, por ende se filtran también aquellas facturas canceladas.

2.3 Selección de parámetros, elbow selection

El método de selección de parámetros elbow es una técnica ampliamente utilizada para determinar el número óptimo de clusters en algoritmos de clustering. Este enfoque se basa en la observación de la forma del gráfico de "*elbow*" generado al trazar el número de clústeres frente a una medida de rendimiento del modelo, como el error intra-cluster. El objetivo es identificar el punto en el que el incremento de clústeres deja de proporcionar mejoras significativas en la estructura de los datos. En este contexto, se ha implementado dicho algoritmo que automatiza la selección del número óptimo de clústeres.

Una de las ventajas de este método es que no requiere conocimiento previo sobre los datos y puede aplicarse a diferentes algoritmos de clustering. Junto a ello, es un método simple y efectivo en la tarea de selección.

Respecto a la implementación, se ha implementado en una función denominada ***elbowSelection***. Este algoritmo toma como entrada una secuencia de costes, que representan los errores intra-cluster para diferentes valores de k, y un ratio umbral para seleccionar el punto de codo. El algoritmo se basa en iterar sobre los costos desde el segundo elemento hasta el último. En la iteración se calcula el ratio de error dividiendo

el costo actual entre el costo anterior (`cost(i) / cost(i-1)`). En el caso que se encuentre un ratio de error mayor al umbral (especificado en los parámetros de la función), se considera que se ha encontrado el punto óptimo y se devuelve el número de clústeres correspondiente (i - 1). Si no se encuentra ningún punto óptimo después de iterar sobre todos los costos, se devuelve el máximo valor de k (es decir, `costs.length - 1`) como el número óptimo de clústeres.

2.4 Entrenamiento - Ejecución

El entrenamiento de los modelos se realiza mediante dos scripts: "***train***" y "***trainBisect***", que serán ejecutados por el programa encargado del entrenamiento. Ambos scripts son prácticamente idénticos, ya que lanzan un pipeline de entrenamiento que incluye la selección y filtrado de características, así como el propio entrenamiento de los modelos. Como resultado del entrenamiento, se obtienen los modelos finales y los umbrales correspondientes.

Para ejecutar estos scripts, se ha creado un script en Bash que los invoca (***start_training***). En este script se especifica el conjunto de datos utilizado para el entrenamiento de los modelos, así como las rutas donde se guardarán los modelos y los umbrales resultantes.

Una vez completada la ejecución del entrenamiento, se obtienen los modelos KMeans (almacenados en el directorio "*/clustering/*") y BisectionKMeans (almacenados en el directorio "*/clustering_bisect/*"). Además de los modelos, se generan dos archivos: "*threshold*" y "*threshold_bisect*". Estos archivos contienen los umbrales obtenidos durante el entrenamiento.

Los umbrales resultantes del entrenamiento son los siguientes:

- **Umbral del modelo KMeans**: 55403.623089850604
- **Umbral del modelo BisectionKMeans**: 58023.35140518957

Estos umbrales son utilizados posteriormente en el proceso de detección de anomalías en el flujo de facturas.

3 Pipeline

En el centro del sistema planteado se encuentra el pipeline de procesamiento y análisis de los datos. En él se define la estructura por la que se recibe el flujo continuo de datos y se procesa para posteriormente enviarlo por distintos tópicos en función del resultado obtenido. El pipeline se encuentra en el paquete *es.dmr.uimp.realtime* y puede ser ejecutado mediante el script ***start_pipeline.sh***. El código se ha desarrollado en dos scripts, ***PipelineFunctions*** y ***InvoicePipeline***.

- **PipelineFunctions**, contiene las funciones implementadas que darán funcionalidad al pipeline diseñado. Se pueden encontrar funciones como la de actualización del estado o filtrado de facturas inválidas.

- **InvoicePipeline**, parte principal del proyecto donde se define el pipeline y se realizan las conexiones pertinentes con los tópicos de Kafka y el contexto de Spark. Se emplean las funciones definidas en *PipelineFunctions*.

3.1 Tipos de datos, Purchases y Invoices

Más allá de los tipos de datos comunes, dado que en este caso se trabaja con el concepto de purchases (compras) e invoices (facturas), se han definido dos clases mediante las cuales representar ambos conceptos en el sistema. En el caso de las compras se define la clase Purchase, esta dispone de una serie de atributos descriptores propios de un elemento "compra". Los atributos de la clase Purchase son:

- **InvoiceNo**: identificador de la factura a la que pertenece la compra.
- **Quantity**: cantidad de elementos que se han comprado.
- **InvoiceDate**: fecha en la que se ha realizado la transacción.
- **UnitPrice**: el precio del producto comprado.
- **CustomerId**: identificador del cliente que ha realizado la transacción.
- **Country**: país donde se ha realizado la transacción.

En el caso de las facturas, de forma similar a las compras, se ha definido una clase, Invoice, la cual dispone de una serie de atributos descriptores. Los atributos de la clase Invoice son:

- **InvoiceNo**: identificador de la factura
- **AvgUnitPrice**: precio del producto medio de la factura.
- **MinUnitPrice**: precio del producto mínimo de la factura.
- **MaxUnitPrice**: precio del producto máximo de la factura.
- **Time**: hora del día de la factura.
- **NumberItems**: número total de productos comprados.
- **LastUpdated**: timestamp de la última modificación de la factura.
- **Lines**: número total de compras pertenecientes a la factura.
- **CustomerId**: identificador del cliente que ha realizado la transacción.
- **State**: estado en el que se encuentra la factura, este puede ser "no emitido", "emitiendo" y "emitido".

Destacar en el caso de Invoice, se ha añadido el atributo "*state*" que junto al atributo "*lastUpdated*" serán de gran utilidad para el manejo de las facturas. En combinación, estos dos atributos nos permiten gestionar que facturas analizar o cuando dejar de recordarlas.

3.2 Gestión del feed de compras

El pipeline comienza estableciendo las conexiones necesarias, como obtener el Spark Context y el Streaming Context. El primer paso es conectarse al feed de compras, lo cual se logra mediante la función ***connectToPurchases***, que ya está disponible en el

proyecto. Esta función lee el feed de compras y devuelve un DStream con una tupla de cadenas de texto.

A continuación, los datos entrantes se procesan para transformar la tupla de cadenas de texto en una tupla que contenga el número de factura (invoiceNo) y un objeto de tipo Purchase. De esta manera, se obtiene un DStream del tipo DStream[(String, Purchase)].

Una vez que los datos entrantes se han transformado en un formato adecuado para nuestro pipeline, es el momento de definir el estado del pipeline, el estado de las facturas. Esto se logra aplicando el método ***updateStateByKey***, el cual recibe la función updateInvoice que hemos definido para controlar y actualizar el estado. El resultado es un DStream del tipo DStream[(String, Invoice)], lo cual nos permite completar las tareas definidas en el indicio del trabajo, como la detección de facturas canceladas, el filtrado de facturas inválidas y el clustering utilizando los modelos pre-entrenados.

Con esta etapa completada, el pipeline está listo para realizar las siguientes acciones sobre las facturas y continuar con las tareas definidas en el trabajo.

3.3 Estado de facturas

Dado que el clustering se hace sobre facturas, mientras que el feed consta de compras individuales. Esto hace que en el pipeline de streaming tenga que mantener un estado con la factura actual y, en base al SLA y la última fecha de modificación de la factura decidir cuándo emitir la factura a la salida y también eliminarla del estado para que este no crezca continuamente. Para lograr esto, se han diseñado una serie de funciones que gestionan el estado y manejan los distintos flujos de datos disponibles.

Para llevar a cabo esta tarea, se han definido un total de tres funciones auxiliares y una función principal llamada "***updateInvoice***". La función "*updateInvoice*" es donde se desarrolla la lógica para manejar el estado. Recibe dos parámetros: "newPurchases", que es una secuencia de nuevas compras que comparten el mismo ID de factura, y "runningInvoice", que representa el estado anterior y puede ser un objeto de tipo factura o nulo en el caso del primer estado (estado inicial).

En cuanto a la lógica de la función, se puede dividir en varias secciones condicionadas por el estado actual de la factura. A continuación, se enumeran las distintas fases de la lógica de actualización en orden de ejecución:

(1) Estado 0

Cuando se realiza la primera actualización, no existe un estado previo, por lo que se genera una nueva factura en función de las compras recibidas como parámetros en la función de actualización. El proceso de construcción y definición de la nueva factura se realiza mediante la función auxiliar "newInvoice". Esta función toma como parámetro la secuencia de nuevas compras ("newPurchases") y devuelve un objeto de tipo factura. Es importante destacar aspectos como el cálculo del precio medio, mínimo y máximo, así como la definición del estado de la factura como "No emitido". Si se

cumple esta condición, la función devolverá el objeto factura creado; de lo contrario, se avanzará a la siguiente fase de actualización.

(2) Facturas emitidas cuya última actualización sea mayor al umbral de tiempo máximo

En caso de que la factura guardada ya haya sido emitida (se verifica a través del atributo "state" del objeto Invoice), se realizan comprobaciones de tiempo para determinar si se debe mantener o eliminar la factura del estado. Se establece un umbral de tiempo máximo para las facturas y se comprueba si el tiempo transcurrido desde la última actualización de la factura hasta el momento actual supera ese umbral. Si se supera, la factura se elimina. En este trabajo, se ha definido un umbral máximo de 8 minutos. Por lo tanto, las facturas con un tiempo de vida superior a 8 minutos y que ya hayan sido emitidas se eliminarán del estado. Si no se cumplen estas condiciones, se avanza a la siguiente fase de actualización y se devuelve el nuevo estado sin la factura analizada.

(3) Facturas en estado “emitiendo”

Cuando la factura guardada en el estado tiene el estado "emitiendo", significa que en la iteración anterior del pipeline se ha procesado. En esta fase, se actualiza el estado de la factura a "emitida". Si no se cumple esta condición, se avanza a la siguiente fase de actualización.

(4) Facturas no emitidas cuya última actualización sea mayor al umbral de tiempo mínimo

En esta fase, se verifica el estado previo de la factura es “no emitido” y se comprueba cuánto tiempo ha pasado desde la última actualización. Se calcula el tiempo transcurrido y se compara con el umbral de tiempo mínimo establecido. Si se supera dicho umbral, se actualiza el estado de la factura a "emitiendo"; de lo contrario, se avanza a la siguiente fase.

(5) Actualización de la factura en base a las nuevas compras

Después de verificar las fases anteriores, se comprueba si existen compras en el parámetro "newPurchases". En caso afirmativo, se actualiza la factura en función de las nuevas compras. Si no hay nuevas compras que añadir a la factura, se pasa a la siguiente fase por defecto. Es importante destacar la parte de actualización de los nuevos valores de la factura, que se realiza mediante la función "updateValuesInvoice". Esta función toma como parámetros las nuevas compras y la factura del estado previo, devolviendo una nueva factura con los valores actualizados.

(6) Fase por defecto

Si no se cumpliesen las condiciones expuestas se ejecutaría esta fase la cual mantiene el estado previo de la factura.

Como se ha mencionado, el flujo de actualización va verificando las condiciones enumeradas y realiza diferentes acciones según se cumplan o no. Es importante mencionar los estados internos de la factura, que pueden ser "no emitido", "emitiendo" o "emitido". Basándonos en las especificaciones del trabajo, el SLA asegura que no haya más de 40 segundos entre compras de la misma factura. Esto nos permite determinar cuándo una factura está completa y lista para ser procesada, evitando procesar facturas incompletas. En cuanto al descarte de facturas, se ha establecido un umbral de 8 minutos, ya que es el tiempo necesario para el pipeline de recuento de facturas canceladas. Todo esto contribuye a mejorar la escalabilidad y el rendimiento del pipeline.

3.4 Sub-pipeline de facturas inválidas

Uno de los subflujos que se han diseñado e implementado es el subflujo de detección de facturas inválidas. Antes de implementar cualquier elemento, es crucial definir que es una factura invalida, para que así el sistema las identifique e aisle. En este caso, se considera una factura anómala cualquiera que cumpla las siguientes condiciones:

- Alguno de sus atributos sea nulo.
- Cualquiera de sus atributos numéricos (avgUnitPrice,minUnitPrice,maxUnitPrice,numberItems,lines) sea menos o igual a 0 excepto el atributo “time” que se considera anormal cuando sea menor que 0.

La lógica se ha implementado en dos funciones principalmente, una principal que maneja el subflujo (***invalidPipeline***) y otra que realiza la detección sobre si una factura es inválida (isInvalid). La función “invalidPipeline” recibe como parámetros el DStream de facturas y devuelve un DStream adaptado a la salida del tópico de kafka (DStream[(String,String)]). Respecto a la implementación, el subflujo consta de dos fases:

- Se itera el DStream de facturas filtrando aquellas facturas que su estado es “emitido” y la llamada a la función “isInvalid” sea true, devolviendo un DStream de facturas filtrado.
- El DStream resultante de la primera fase se transforma adaptándolo a la salida del topic de Kafka obteniendo un DStream de tuplas de cadenas de texto (DStream[(String,String)])

Finalmente se recorre el DStream resultante publicando cada elemento en el topic “invalid_invoices” de Kafka mediante la función “***publishToKafka***”. La ejecución del flujo se puede encontrar en el script InvoicePipeline, mientras que las funciones, a excepción de publishToKafka, se encuentran declaradas en el script PipelineFunctions. Un ejemplo de la llamada es:

```
// ------------------- INVALID INVOICES -----------------
--
val invalidDStream = invalidPipeline(invoicesDStream) //
Get invalid invoices
invalidDStream.foreachRDD(rdd => publishToKaf-
ka("invalid_invoices")(broadcastBrokers)(rdd))
```

3.5 Sub-pipeline de facturas canceladas

Dentro del flujo de datos recibidos a través del feed, uno de los tipos de datos que se pueden encontrar son las facturas canceladas, las cuales se identifican por su número de factura (invoiceNo) que comienza con la letra "C". Con el objetivo de cumplir con los requisitos del ejercicio, se ha diseñado e implementado un sub-pipeline dedicado al recuento de facturas canceladas en los últimos 8 minutos, con una actualización cada minuto. Siguiendo un enfoque similar al sub-pipeline de facturas inválidas, se ha creado la función ***cancellationPipeline*** para llevar a cabo la lógica del pipeline.

(1) **Fase de filtrado:** En esta etapa, se filtra el DStream de facturas para retener únicamente aquellas facturas cuyo estado no sea "no emitida" y cuyo número de factura (invoiceNo) comience con la letra "C". El resultado de esta acción es un nuevo DStream que contiene únicamente las facturas que cumplen ambas condiciones mencionadas.
(2) **Fase de conteo:** Aplicando el método countByWindow al DStream filtrado, se obtiene un nuevo DStream que contiene el recuento total de facturas canceladas en los últimos minutos, de acuerdo con el tamaño de la ventana especificado.
(3) **Fase de transformación:** El DStream resultante de la fase anterior se transforma en otro DStream que se adapta a la salida requerida por la función. Este nuevo DStream se utilizará posteriormente para publicar los resultados en el topic de Kafka correspondiente.

Por último, el DStream resultante se recorre y cada elemento se publica en el topic "cancelations_ma" de Kafka utilizando la función publishToKafka. El flujo de ejecución se puede encontrar en el script "InvoicePipeline", mientras que las funciones, a excepción de publishToKafka, se declaran en el script "PipelineFunctions". A continuación, se muestra un ejemplo de cómo se realiza la llamada a la función:

```
 // ------------------- CANCELED INVOICES ---------------
 // Define window and slide interval
 val WINDOW_LENGTH = 8 // 8 minutes
 val SLIDE_INTERVAL = 60 // 1 minute
// Get cancelations in the last 8 minutes every 1 minute
val cancelDStream = cancellationPipeline(invoicesDStream,
WINDOW_LENGTH, SLIDE_INTERVAL)     cancelD-
Stream.foreachRDD(rdd => publishToKaf-
ka("cancelations_ma")(broadcastBrokers)(rdd))
```

Es importante destacar el uso de la función countByWindow en este sub-pipeline, ya que permite realizar el conteo de facturas canceladas en un período de tiempo específico utilizando una ventana deslizante. Esta función proporciona una forma conveniente de calcular automáticamente el recuento dentro de la ventana deslizante, sin necesidad de implementar manualmente la lógica de conteo. Además, al utilizar una ventana deslizante, se obtienen recuentos actualizados en intervalos regulares a medida que se procesa el flujo de facturas entrante.

Un ejemplo de la salida obtenida en el topic "cancelations_ma" de Kafka sería el siguiente:

```
bigdata@bigdata:/opt/Kafka/kafka_2.11-2.3.0$
bin/kafka-console-consumer.sh --bootstrap-server lo-
calhost:9092 --topic cancelations_ma
Facturas canceladas en 1 minutos: 52
Facturas canceladas en 1 minutos: 138
Facturas canceladas en 1 minutos: 249
Facturas canceladas en 1 minutos: 315
Facturas canceladas en 1 minutos: 348
Facturas canceladas en 1 minutos: 363
Facturas canceladas en 1 minutos: 399
Facturas canceladas en 1 minutos: 444
Facturas canceladas en 1 minutos: 501
```

En este ejemplo, se ha especificado un tamaño de ventana de 1 minuto con un intervalo de desplazamiento de 20 segundos.

3.6 Sub-pipeline de clustering mediante KMeans y BisectionKMeans

Por último, se encuentra el sub pipeline donde se detectan facturas anómalas aplicando los modelos de clustering entrenados con anterioridad. Como se indica en las secciones anteriores, estos modelos son KMeans y BisectionKMeans y tratrán de detectar las facturas anómalas que se encuentren en el conjunto de datos. Dentro del script "InvoicePipeline", se realizan las siguientes etapas para el sub-pipeline de clustering mediante KMeans y BisectionKMeans:

(1) **Carga de modelos y umbrales:** Se cargan los modelos de clustering y los umbrales necesarios utilizando las funciones loadKMeansAndThreshold y loadBisectKMeansAndThreshold. Estos modelos y umbrales se utilizan posteriormente en el proceso de clustering.
(2) **Transmisión de umbrales**: Los umbrales necesarios se transmiten como variables de difusión (variables broadcast) para que estén disponibles en todos los nodos del clúster.
(3) **Clustering utilizando KMeans**: Se aplica el clustering utilizando el modelo de KMeans y el umbral correspondiente. Esto se realiza llamando a la

función ***clusteringPipeline*** con el DStream de facturas, el modelo de KMeans y el umbral transmitido. El resultado del clustering se publica en el tópico de Kafka "*anomalies_kmeans*".

(4) **Clustering utilizando BisectionKMeans**: De manera similar al paso anterior, se realiza el clustering utilizando el modelo de BisectionKMeans y su umbral correspondiente. Se llama a la función *clusteringPipeline* con los parámetros adecuados y el resultado se publica en el tópico de Kafka "*anomalies_kmeans_bisect*".

La función ***clusteringPipeline*** se encarga de procesar el flujo de facturas y realizar el clustering para identificar las facturas anómalas. La lógica detrás de la función es:

- **Filtrado de facturas**: Se filtran las facturas en estado "emitiendo" del DStream de facturas.
- **Cálculo de distancias**: Para cada factura en estado "emitiendo", se selecciona un conjunto de atributos (precio del producto medio, precio del producto mínimo, precio del producto máximo, hora del día de la factura y número total de productos comprados). Luego, se realiza la predicción del cluster correspondiente utilizando el modelo adecuado (KMeans o BisectionKMeans) y se obtiene el centroide del cluster. A continuación, se calcula la distancia entre la factura y el centroide utilizando la función sqdist de vectores.
- **Filtrado de facturas anómalas**: Se filtran las facturas que superan el umbral de distancia establecido como anómalas.
- **Transformación para publicación**: Se transforma el resultado para que pueda ser publicado en el topic de Kafka correspondiente. Esto implica generar un mensaje informativo que describe la factura anómala, incluyendo su número de factura y la distancia calculada.

De esta manera, el sub-pipeline de clustering mediante KMeans y BisectionKMeans permite identificar y publicar facturas anómalas basándose en la distancia a los centroides de los clusters correspondientes.