Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección de anomalías en eventos de seguridad en tiempo real

Autor: Juan Jesús Chorro Bergillo
Tutor: Pablo Nebrera Herrera

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección de anomalías en eventos de seguridad en tiempo real

Autor:
Juan Jesús Chorro Bergillo

Tutor:
Pablo Nebrera Herrera
Profesor asociado

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Grado: Detección de anomalías en eventos de seguridad en tiempo real

Autor: Juan Jesús Chorro Bergillo

Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mis profesores, por mostrarme el camino a seguir

A mi familia, por ayudarme a caminarlo

A Cristina, por escuchar cada metro recorrido

Agradecimientos

Con estas palabras describo lo que es el último escalón a subir de esta gran etapa de mi vida, han sido muchas las personas que me he encontrado en el camino y todas han aportado su granito de arena.

Primero de todo agradecer a cada uno de los miembros de mi familia, los que están y los que estuvieron. Gracias a mis padres, sobretodo, por el esfuerzo que hacían día a día para yo tener esta oportunidad y por los ánimos que me dieron a lo largo de mi camino, ellos son los que me han dado las alas para moverme libremente por este mundo.

Tampoco se olvidarán aquellos ratos de estudios y de ocio con mis compañeros de clase, grandes personas donde las haya, siempre disponibles cuando los necesitas. Gracias, porque vosotros también me habéis enseñado a apreciar el trabajo en equipo y aprender de nuestros errores.

También dar las gracias a la empresa de ENEO tecnologías, a Pablo y Jaime Nebrera y por supuesto al equipo de trabajo que me brindaron la oportunidad, no sólo de poder hacer este proyecto, sino además de emprenderme en el mundo laboral dando a conocer mis habilidades y aptitudes.

Como en toda buena comida no puede faltar un buen postre, Cristina, gracias a ti en especial por creer en mí, por los ánimos y por escuchar cada batalla librada en este proyecto.

Juan Jesús Chorro Bergillo

Grado en Ingeniería de las Tecnologías de Telecomunicación

Sevilla, 2015

Resumen

En este proyecto se pretende integrar un detector de anomalías dentro de una arquitectura Big data para la monitorización de eventos de seguridad y enriquecerlos con los resultados obtenidos tras los análisis. Para ello entraremos en el uso y comprensión de las tecnologías Big data actuales de la mano del proyecto Horama de redBorder. Desde la captura de los eventos de seguridad, su distribución y almacenado, hasta llegar al análisis de los eventos para la detección de anomalías, análisis de resultados y enriquecimiento.

Para llevar a cabo todo el proyecto se han empleado herramientas de Apache como Kafka para la distribución y el soporte de la llegada masiva de eventos, ZooKeeper para la coordinación y sincronización de Kafka así como Hadoop para la asignación de recursos a través de YARN y empleo de Samza para la ejecución de las aplicaciones que tratarán los flujos de entrada definido. Analizaremos numerosas implementaciones para el análisis de los eventos y finalmente nos decantaremos por un framework que incluye una colección de librerías de machine learning para el tratamiento de flujos de datos evolutivos denominado MOA.

Abstract

This project aims to integrate a outlier detector within an architecture Big data for monitoring security events and enrich them with the resultos obtained from the analysis. To do this go into the use and understanding of Big data technology current hand Horama redBorder project. From capturing security events, distributed and stored, until the analysis of events for anomaly detection, analysis of results and enrichment.

To carry out the entire project have been used as tools of Apache Kafka for the distribution and support of the massive arrival of events, ZooKeeper for coordination and synchronization of Kafka and Hadoop for the allocation of resources through YARN and employment Samza for the implementation of applications that seek input flows defined. We analyze numerous implementations fro analyzing events and finally we choose for a framework that includes a collection of machine learning libraries for the treatment of developmental data streams called MOA

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

"Estudia el pasado si quieres pronosticar el futuro"

- Confucio -

Es evidente, desde el punto de vista tecnológico, que los avances son cada vez mayores. Tenemos una tecnología cada vez más rápida y útil, que no sólo aporta comodidades a nuestra vida por su funcionalidad, sino que además aportan información, una gran cantidad de información.

Un claro ejemplo de esta evolución es conocido como el internet de las cosas (IoT), donde en un futuro tendremos la capacidad de interconectar con nuestro entorno ofreciéndonos una nueva realidad, sin embargo, esto significa que se generará aún más información y ésta deberá ser analizada y tratada empleando para ello la estadística o algoritmos de machine learning.

En este breve capítulo daremos una introducción a la era de la información en la cual vivimos y cuáles han sido las motivaciones que nos ha impulsado a desarrollar este proyecto así cómo definir los objetivos a cumplir.

1.1 La era de la información

El crecimiento de la información es cada vez mayor, si observamos el gráfico inferior podemos observar que el término "Big Data" se ha convertido en una de las tendencias de búsqueda en Google en los últimos años.

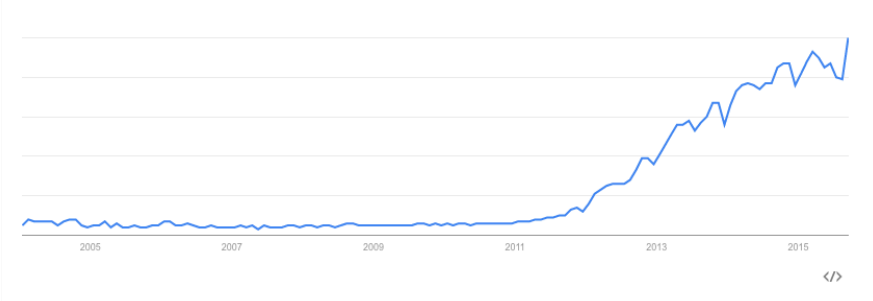

Figura 1-1. Tendencia del término "Big data". Fuente : *www.google.com/trends*

Esto ha propiciado el uso de muchas técnicas de estadísticas y de machine learning para su análisis y tratamiento y así encontrar patrones que permitan conocer las características de los usuarios, la predicción de los sistemas e incluso la detección de anomalías.

1.1.1 Big Data

Big data es el término acuñado a los grande volúmenes de información generados por los usuarios o dispositivos de forma continua. El gran volumen de información, su variedad y su velocidad es lo que caracteriza a este término. Conocido como las 3Vs, la definición de Big data puede entenderse de la siguiente forma:

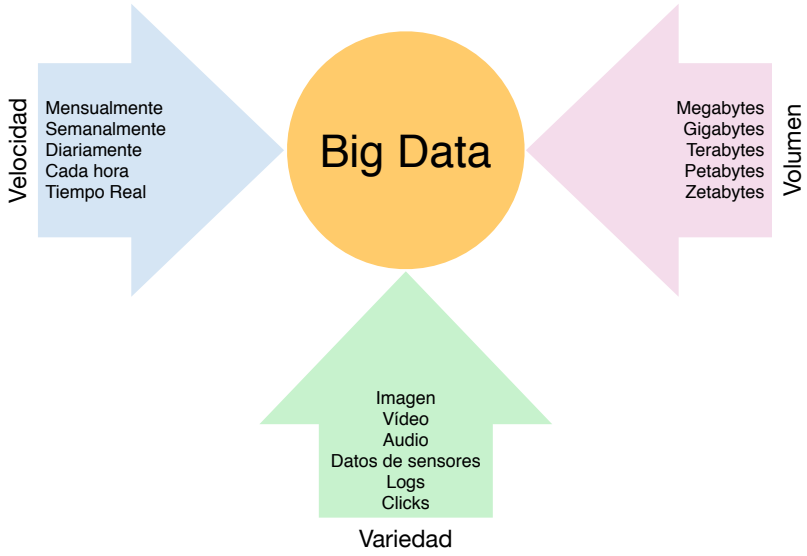

Figura 1-2. Definición de "Big Data" a partir de las 3Vs

- **Velocidad :** Los medios sociales han contribuido a la vertiginosa velocidad con la que son enviado los datos, cada día 4500 fotos son subidas a Facebook en un segundo, se envían más de 100.000 tweets por segundo y se suben más de 40 horas de vídeos en youtube en un minuto.
- **Volumen :** Vemos junto con la velocidad un crecimiento exponencial en el almacenamiento de los datos ahora que son algo más que un simple texto. Podemos encontrar datos de vídeo, imagen y sonido en nuestros medios sociales, por lo que no es de extrañar encontrar Terabytes y Petabytes de información en las grandes empresas.
- **Variedad :** No todos los datos son iguales y es por ello que deben de tener un formato acorde a su finalidad, como un vídeo en mp4, una foto en jpeg o el número de clicks en una página web.

Como podemos apreciar es necesario adoptar una serie de herramientas que permitan lidiar con todas las características que conlleva el término Big data. En capítulos posteriores introduciremos dichas herramientas y como contribuyen en el desarrollo del proyecto.

1.1.2 Machine Learning

Machine learning es una rama de la inteligencia artifial. Usando la computación podemos diseñar sistemas que pueden aprenden de los datos a modo de entrenamiento. Los sistemas podrían aprender y mejorar con la experiencia y, con el tiempo, refinar los modelos que pueden ser usados para predecir respuestas a cuestiones

basadas en lo aprendido.

Podemos distinguir dos tipos de aprendizajes:

- **Supervisado :** Se refiere a trabajar con un conjunto de datos de entrenamientos etiquetados. Para cada muestra de los datos de entrenamientos tienes una entrada y una salida esperada. Las regresiones lineales y polinomiales son ejemplos de este tipo de entrenamiento
- **No supervisado :** Donde los algoritmos encuentran patrones ocultos en los datos. Con el aprendizaje no supervisado no hay respuesta correcta o incorrecta. Los detectores de anomalías, la clusterización y la clasificación son ejemplos de este tipo de entrenamiento.

Y del mismo modo dos tipos de procesamientos de datos:

- **Off line :** Más conocido como procesamiento por lotes (Batch) en el que los datos son almacenados para su posterior tratamiento con los algoritmos de entrenamiento. Como ejemplo tenemos WEKA que contiene un amplio conjunto de algoritmos de aprendizaje para este fin.
- **On line :** Conocido como stream, en el que los algoritmos van aprendiendo en tiempo real, permitiendo definir modelos adaptados al presente. Aquí es donde nos encontramos con MOA que posee un conjunto de algoritmos para este tipo de entrenamiento.

1.2 Motivación

Para la detección de anomalía en los eventos de seguridad es necesario manipular la gran cantidad de información que suponen dichos eventos y para su procesado es necesario el uso de las herramientas de machine learning adecuadas.

Los eventos deben ser tratados en tiempo real, es por ello que hay que utilizar algoritmos diseñados para un procesamiento rápido y efectivo.

Por un lado encontramos el famoso workbench WEKA, utilizado para el análisis del conocimiento. WEKA está escrito en Java y su procesamiento de datos es en batch. Ofrece una gran cantidad de algoritmos a utilizar para el análisis de los datos y recientemente han trabajado junto con el equipo de MOA para ofrecer una alternativa en flujos de datos. WEKA es centralizado y por lo tanto no podemos ejecutarlo de forma concurrente.

El equivalente a WEKA pero de forma distribuida es MAHOUT. El objetivo de este entorno es ofrecer aplicaciones de machine learning escalables. Al hablar de escalabilidad eliminamos la barrera de un único punto de fallo, sin embargo, el procesamiento de datos de MAHOUT sigue siendo por lotes, lo que no es una alternativa adecuada para el propósito de este proyecto.

Antes cuando hablamos de WEKA, mencionamos el proyecto MOA. MOA es un framework para el análisis de flujos de datos evolutivos en tiempo real. Ofrece una gran variedad de algoritmos entre los que encontramos los de detección de anomalías. Al igual que WEKA, está escrito en Java lo que nos da la posibilidad de portarlo a otras plataformas. MOA también es centralizado, y es una clara desventaja para lo que necesitamos, sin embargo, intentaremos potenciar su funcionalidad con Samza.

Al igual que MAHOUT es para WEKA su versión distribuida, MOA tiene su homólogo y se conoce como SAMOA. SAMOA es un reciente proyecto de Apache, por lo que aún no existe una versión estable a día de hoy. SAMOA también procesa los flujos en tiempo real y para ello utiliza algoritmos distribuidos. Sin embargo, dentro de estos algoritmos no están implementados aún los de detección de anomalías, así que aunque se trata de una buena alternativa, no podemos hacer uso de ello.

De forma resumida nos encontramos con la siguiente comparativa donde en la parte superior se especifica el tipo de aprendizaje y en la parte izquierda si es centralizado o distribuido:

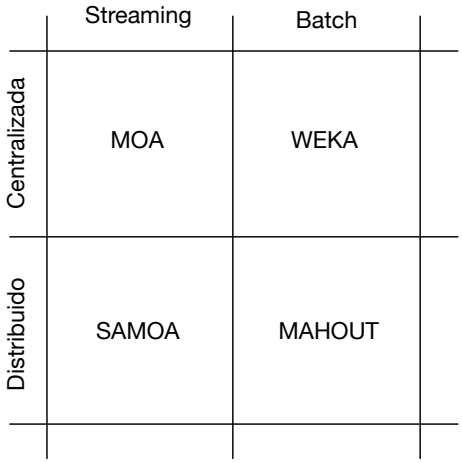

Figura 1-3. Comparativa de los distintos casos de estudio

Como podemos observar disponemos de un marco de opciones variadas para llevar a cabo el proyecto, lo ideal es utilizar SAMOA como se mencionó anteriormete para emplear algoritmos distribuidos, sin embargo el proyecto se encuentra en una fase muy temprana y aún no ofrece la funcionalidad de detección de anomalía de forma distribuida. Dada las necesidades que tenemos, nos decantaremos por MOA.

Aunque MOA es centralizado contiene las funcionalidades que necesitamos que es el análisis y tratamiento de flujo de datos evolutivos. Intentaremos potenciar su funcionalidad utilizando Samza y evaluaremos los resultados obtenidos para ver la viabilidad de implementar este framework dentro de Horama.

1.3 **Objetivos**

Hemos hecho un pequeño análisis para tener una visión global del problema, disponemos de muchos datos y de varias técnicas para tratarlos, la vertiente del proyecto dependerá por lo tanto de cumplir los siguientes objetivos:

- Analizar y entender las herramientas de Apache orientadas a Big data.
- Analizar y entender el funcionamiento de MOA así como las capacidades que posee.
- Desarrollar e implementar una tarea de Samza que permita integrar a MOA en el entorno distribuido y comprobar su viabilidad.

1.4 Trayectoria

En el siguiente capítulo nos centraremos en la arquitectura del proyecto, en él se relatará de forma breve los distintos componentes que permiten la recolección y administración de los eventos que se están recibiendo. Esto es fundamental ya que simularemos los flujos de eventos para llevar a cabo las pruebas necesarias y así poder integrarlo posteriormente en Horama.

A continuación pasaremos a analizar MOA y entraremos en profundidad sobre el concepto de detección de anomalía y veremos cuál es el que implementa, daremos algunas pinceladas de los algoritmos más utilizados y finalmente elegiremos uno para la elaboración de la tarea de Samza basándonos en el tiempo de procesamiento, ya que necesitamos eficiencia y rapidez.

Por último mostraremos la lógica detrás de las tareas de Samza y en ella integraremos el algoritmo de MOA, pondremos en marcha el sistema completo, generaremos los eventos sintéticos y comprobaremos los resultados obtenidos tras ello.

Con esta trayectoria podremos explorar cada uno de los objetivos y llegar a la conclusión final sobre la viabilidad de MOA.

2 ARQUITECTURA DE SOPORTE

En este capítulo hablaremos de la arquitectura necesaria para el almacenamiento, distribución y procesado de los grandes volúmenes de datos. Para ello vamos a introducir los conceptos básicos sobre las herramientas de Apache que permiten realizar dichas tareas. Estas herramientas son ZooKeeper, Kafka, Hadoop y Samza.

Hay que tener en cuenta que elaborar una arquitectura sólida es vital, pues necesitamos una alta disponibilidad de los datos que se están recibiendo y una buena coordinación entre las aplicaciones para evitar retrasos e incoherencia en nuestros datos. Además, la producción sintética de eventos y su comportamiento debe ser lo más fiel posible al escenario planteado por redBorder.

2.1 Arquitectura

Podemos resumir la arquitectura necesaria para nuestro proyecto en la siguiente figura. En ella podemos ver varias herramientas de Apache, como ZooKeeper encargado de la coordinación de los brokers de Kafka, Hadoop que junto con YARN administra aplicaciones distribuidas y gestiona los recursos que aprovechará Samza para ejecutar sus tareas y dentro de una de ellas MOA.

Podemos distinguir también dentro de la figura el productor sintético, este productor es parte de uno de los proyectos de redBorder, utilizado en la fase de desarrollo de sus productos y el cual está encargado de producir eventos de forma sintética, permitiendo una fácil y personalizada configuración sobre los datos a generar. Este factor es importante, ya que nos dará la posibilidad de simular rangos de IP, conjuntos de páginas web y protocolos de red de una forma muy sencilla.

Por supuesto es necesario no solo el análisis de los datos obtenidos, sino también un registro de estos por si el sistema fallase o por si quisieramos consultar un histórico. Es por ello que también la tarea de Samza encargada de ejecutar MOA generará un log donde se registrará toda la actividad de dicha tarea así como los resultados obtenidos.

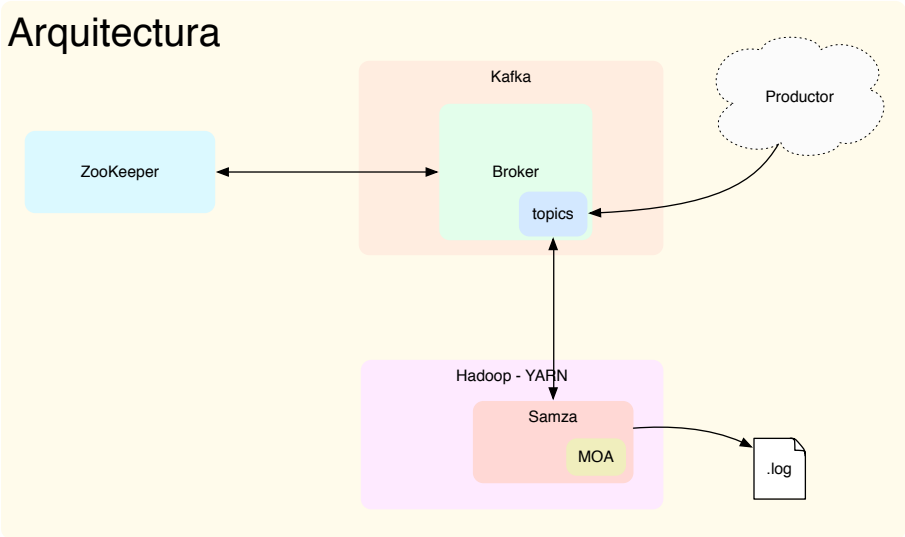

Figura 2-1. Arquitectura para la el tratamiento y distribución de los eventos

Daremos una breve introducción a cada uno de los componentes así cómo detallar algunas funcionalidades que hacen que sean útiles en este proyecto y cómo influyen en él.

2.2 ZooKeeper

ZooKeeper fue diseñado para ser un robusto servicio que permitiría a los desarrolladores centrase principalmente en la lógica de sus aplicaciones antes que en la coordinación de éstas. Expone una sencilla API, inspirada en un sistema de ficheros, que permite a los desarrolladores implementar tareas comunes de coordinación. Ante un conjunto de servidores, ZooKeeper es tolerante a fallos y, lo más deseable, escalable.

ZooKeeper es necesario para el funcionamiento de Kafka ya que la sincronización de los brokers y los topics son llevados por él, así como la coordinación de Hadoop dentro de los clústeres formados.

2.2.1 Fundamentos básicos

Como se comentó anteriormente ZooKeeper proporciona una API, semejante a la de un sistema de ficheros, compuesta de un pequeño conjunto de llamadas que permite a las aplicaciones implementar sus propias primitivas. A estas primitivas se las llamará recetas para denotar la implementación de dichas primitivas. Las recetas incluyen operaciones de ZooKeeper que manipulan pequeños nodos de datos llamados znode que están organizados en un árbol jerárquico, como si de un sistema de ficheros se tratase.

```
[zk: localhost:2181(CONNECTED) 1] ls /
[controller_epoch, controller, brokers, zookeeper, admin, consumers, config]
```

Figura 2-2. Ejemplo de la coordinación de ZooKeeper

Los znodes pueden ser de diversos tipos con características particulares:

- **Persistentes**: Un znode es persistente cuando la ruta puede ser únicamente eliminada a través de una llamada a la API de ZooKeeper. Esto significa también que si un cliente es desconectado la referencia al nodo seguirá intacta.
- **Efímeros**: Un znode es efímero cuando la ruta es eliminada si el cliente que lo crea cae o si se pierde la conexión con ZooKeeper. Esto es útil para eliminar referencias que ocupan espacio con el tiempo, es importante tener en cuenta esto si en ellas se almacena información de algún tipo.
- **Secuenciales**: Un znode secuencial tiene asignado un único número entero incremental que se añade al final de la ruta. Es una forma interesante de conocer el número de veces que se ha levantado un znodo, por ejemplo.

Lo más normal es que ZooKeeper sea accedido de forma remota como un servicio. Cada vez que un cliente necesite conocer el contenido de un znode debe realizar una petición con la primitiva adecuada, pero desembocará en una latencia que podría ser elevada, este problema normalmente está relacionado con el polling y está relacionado con la espera activa, dado que se están mandando continuamente peticiones cuya respuestas son redundantes e innecesarias.

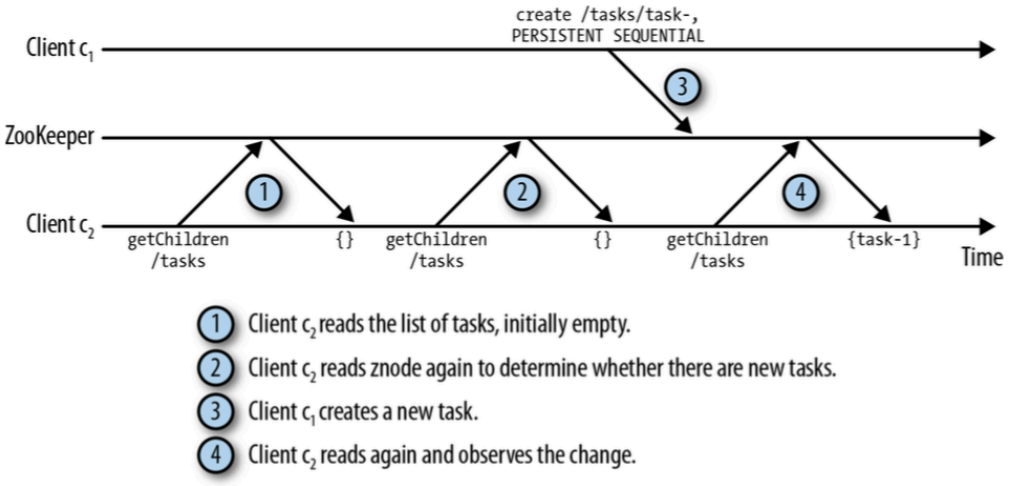

Figura 2-3. Polling en ZooKeeper. Fuente : *ZooKeeper. Distributed process coordination*

Como podemos observar en la imagen C_2 está efectuando peticiones de forma continua a ZooKeeper solicitando los znodes hijos que cuelgan de /task, sin embargo ZooKeeper responde con el mismo resultado una y otra vez, que es un vector vacío (dado que no existe aún un hijo que cuelgue de /task). Supongamos que C_1 crea un nuevo znode persistente y secuencial en un tiempo t. En la siguiente vez que C_2 realice la petición de los hijos de /task, ZooKeeper contestará con un resultado totalmente distinto y satisfaciendo la solicitud.

Estas peticiones que consumen tiempo de procesado sin resultado aparente es un problema muy común del polling. Para su reemplazo ZooKeeper implementa un mecanismo basado en notificaciones. Basicamente un cliente es notificado cuando ZooKeeper descubre un cambio en una ruta, para ello es necesario registrarse como un watch. Un watch es una operación de un único disparo, de manera que lanza una única notificación.

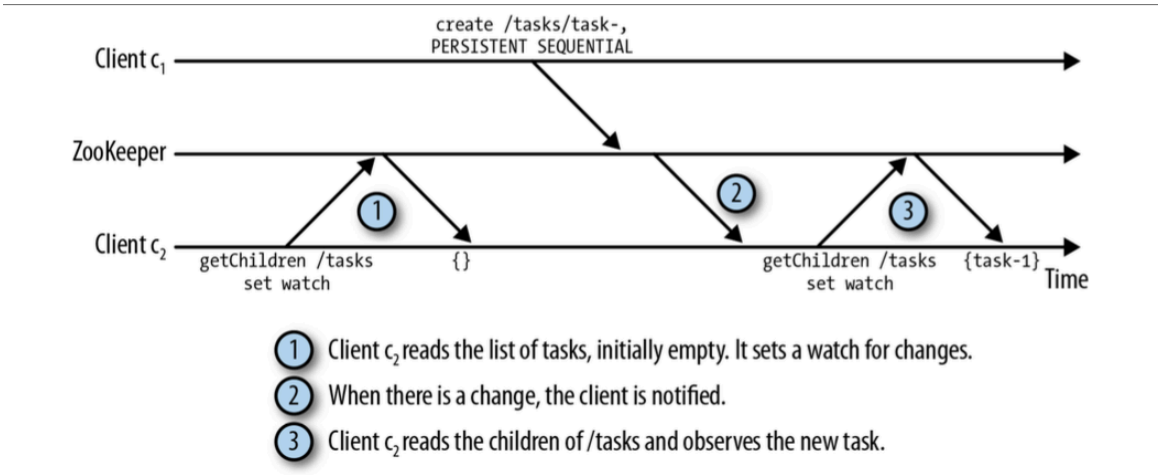

Figura 2-4. Notificaciones en ZooKeeper. Fuente : *ZooKeeper. Distributed process coordination*

Si observamos la imagen podemos ver el potencial de esta implementación, el cliente C_2 solicita los znode hijos de /task a ZooKeeper y a su vez establece un nuevo watch. ZooKeeper devuelve resultados que no aportan ninguna información (el vector está vacío). Cuando en un instante t, C_1 crea un nuevo znode dentro de /task persistente y secuencial éste es registrado en las rutas de ZooKeeper y éste al detectar el cambio se lo notifica a C_2 de manera que al recibir la notificación efectúa nuevamente la petición para obtener los znode hijos de /task respondiendo ZooKeeper, ahora sí, con el resultado que necesita C_2. Como puede deducirse el tiempo de CPU es menor que el de polling y no sobrecarga de peticiones el servicio.

Podemos hacernos una idea de la utlidad de estas características de ZooKeeper, por ejemplo, el levantamiento de nodo Kafka podría monitorizarse con watch y de esa forma sabríamos en todo momento el número de nodos levantas, del mismo modo puede hacerse para el caso contrario y pedir que se notifique cuando un nodo de Kafka se ha caído. De esta forma tan simple pueden monitorizarse los nodos y tener siempre un control sobre ellos.

Por supuesto esto no es exclusivo de Kafka y puede extenderse a todas aquellas aplicaciones que utilicen ZooKeeper, así como aquellas de desarrollo propio. Una forma muy fácil de utilizar la API de ZooKeeper es utilizar Curator, escrito en Java y con la característica de aprovechar el potencial de ZooKeeper.

2.3 Apache Kafka

Apache Kafka es una solución de mensajería basado en colas por publicación/suscripción pensado para un funcionamiento distribuido en tiempo real, manejando cientos de megabytes de lecturas y escrituras por segundo desde miles de clientes. Los flujos de datos son divididos y repartidos sobre un clúster para permitir flujos de datos más voluminosos cuya capacidad sobrepasa cualquier máquina individual y para coordinar a los consumidores. Los mensajes son almacenados en el disco y replicados en el clúster para evitar la pérdida de información. Cada agente puede manejar terabytes de mensajes sin sufrir un gran impacto en el rendimiento. Las principales características de Kafka son:

- **Mensajería persistente :** Permitiendo el almacenamiento de la información en disco así como la replicación de ésta dentro de un clúster, evitando la pérdida de información y mejorando la disponibilidad.

- **Alto rendimiento :** Con el concepto de Big data en mente, Kafka está diseñado para manipular cientos de MBs de escritura y lectura por segundo para un gran número de clientes.
- **Distribuido :** Apache Kafka con un diseño de clúster central soporta el particionado de los mensajes sobre servicios Kafka y distribuir el consumo sobre un clúster de consumidores. El crecimiento del clúster de Kafka es elástico y transparente sin ningún tipo de retraso.
- **Soporte para varios clientes :** El sistema de Apache Kafka suporta la integración de clientes de distintas plataformas como Java, .NET, PHP, Ruby, y Python.
- **Tiempo real :** De forma que los mensajes producidos por el hilo productor deberán de estar inmediatamente visible para los hilos consumidores. Esta carácterística es crítica para los sistemas basados en eventos.

2.3.1 Elementos de Kafka

Un clúster de Kafka está caracterizado por cinco componentes esenciales que interactú:

- **Topic :** Un topic es una categoría donde los mensajes son publicados por el hilo productor de mensajes. Cada topic de Kafka puede estar particionado. Cada partición estará representada por una secuencia inmutable de mensajes que son continuamente añadidos. Un clúster de Kafka mantiene el registro del particionado de cada topic. Cada mensaje en la partición está asignada a un único ID secuencial llamado offset.

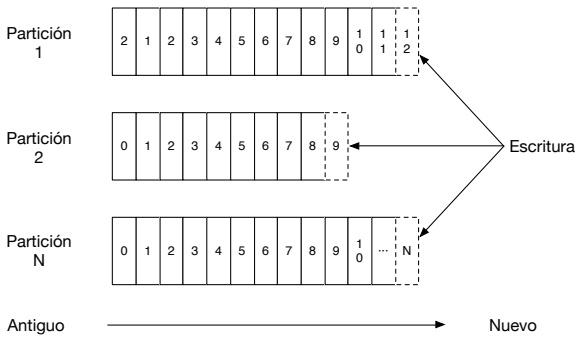

Figura 2-5. Visualización del concepto de particionado en los topics de Kafka

El clúster de Kafka retiene todos los mensajes publicados, tanto si han sido consumidos como si no, por un periodo de tiempo configurable. Por ejemplo, si el registro de retención está fijado para dos días, en esos dos días aquellos mensajes que sean publicados estarán disponibles para su consumo, pasado esos dos días los mensajes serán descartados para la liberación de espacio en el disco. El rendimiento de Kafka es constante y efectivo con respecto al tamaño de los datos así que retener un volumen considerable de datos no supone un problema.

El offset es controlado por el consumidor, normalemente un consumidor avanzará el offset de forma lineal tanto como mensajes sean leídos, pero hay casos en los que la posición es determinada por el consumidor, que puede consumir mensajes en el orden que sea necesario. Por ejemplo, un consumidor puede resetear el offset a un punto anterior para reprocesar los mensajes ya publicados.

Esta combinación de características hace que los consumidores de Kafka sean muy versátiles.

- **Broker :** Un clúster de Kafka consiste en uno o más servidores donde cada uno podría tener uno o más procesos Kafka en ejecución, estos procesos son los llamado brokers. Los topics son creados dentro del contexto que los brokers procesan. De forma que un topic tiene asociado un broker maestro mientras que aquellos a los que se le asigne la partición (si el topic está particionado) son seguidores (followers).

- **Zookeeper :** La función de ZooKeeper es como interfaz de coordinación entre los brokers de Kafka y los hilos consumidores. Además de ello, permite otros usos como el descubrimiento de levantamiento y caída de nodos.

```
[zk: localhost:2181(CONNECTED) 0] ls /
[controller_epoch, brokers, zookeeper, admin, consumers, config]
[zk: localhost:2181(CONNECTED) 1] ls /brokers
[ids, topics]
[zk: localhost:2181(CONNECTED) 2] ls /brokers/topics
[rb_flow]
[zk: localhost:2181(CONNECTED) 3] ls /brokers/topics/rb_flow
[partitions]
[zk: localhost:2181(CONNECTED) 4] ls /brokers/topics/rb_flow/partitions
[0]
```

Figura 2-6. Ejemplo de la coordinación de ZooKeeper para Kafka

- **Productores :** Los hilos productores publican datos en los topics seleccionando la partición adecuada dentro del topic. Los productores pueden publicar cualquier tipo de dato desde cadenas de texto hasta archivos binarios.

- **Consumidores :** Los hilos consumidores son los encargados de recolectar los datos publicados en los topics a los que se suscriben. Es necesario un coordinador ZooKeeper para su coordinación con los topics.

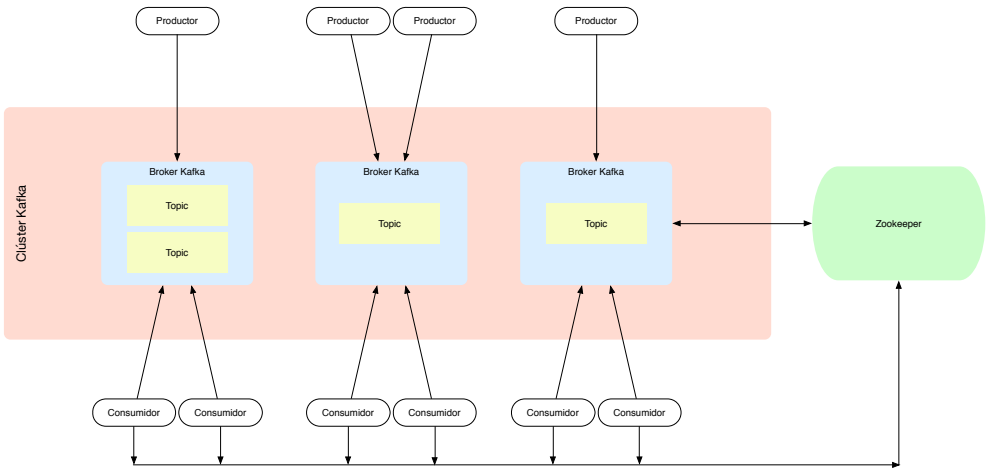

Figura 2-7. Ejemplo de un clúster de Kafka con ZooKeeper como coordinador

En la figura anterior podemos observar un clúster con tres nodos de Kafka, observamos que tanto los brokers como los consumidores se coordinan con ZooKeeper.

2.3.2 Replicación y particionado

En Kafka, la decisión de cómo un mensaje es particionado es llevada a cabo por el productor, el bróker almacena los mensajes en el mismo orden de llegada. El número de particiones pueden ser configurados para cada topic de un bróker Kafka, esto permite distributir los mensajes añadiendo tolerancia ante la carga de grandes volúmenes de información.

La replicación es una de las características más importantes de Kafka. Como mencionamos antes Kafka es altamente escalable, para permitir una mejor durabilidad de los mensajes y una alta disponibilidad de estos en los clúster donde está Kafka, se utiliza la replicación garantizando que los mensajes serán publicados y consumidos incluso si un broker falla por cualquier razón. Tanto el productor como el consumidor tienen la capacidad de replicación en Kafka.

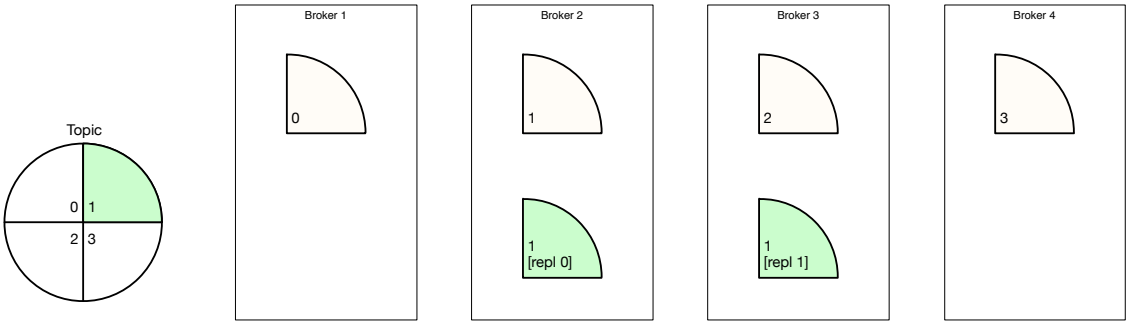

Figura 2-8. Ejemplo gráfico de particionado y replicación de un topic

En la replicación, cada partición de un mensaje tiene n réplicas y pueden permitirse n – 1 fallos para garantizar la entrega de los mensajes. De las n réplicas, una actua como la réplica conductora para el resto de réplicas. Kafka soporta los siguientes modos de replicación

- **Replicación Síncrona :** En este modo, un productor primero identifica la réplica conductora desde Zookeeper y publica el mensaje. Tan pronto como el mensaje es publicado, este es escrito en el log de la replica conductora y todas las que la siguen empiezan a sincronizar los mensajes. Cada replica seguidora envía un asentimiento a la replica conductora una vez que el mensaje es escrito en su respectivo log. Una vez que la replicación se ha completado y todos los asentimientos esperados son recibidos, la replica conductora envía un asentidmiento al productor. Este método es lento, sin embargo asegura la réplica del mensaje.

- **Replicación asíncrona :** La única diferencia con este modo es que, tan pronto como la réplica conductora escribe el mensaje en su log local, esta envía el asentimiento al cliente de mensajes y no espera asentimientos de las replicas seguidoras. Este modo no asegura la entrega de mensajes en el caso de que un broker falle, pero evidentemente es más rápida.

2.4 Hadoop

Apache Hadoop nace con el concepto de una computación fiable, escalable y distribuida. Se trata de un framework que permite el procesamiento distribuido de grandes volúmenes de datos a través de clústeres de ordenadores que utilizan un modelo simple de programación.

A continuación daremos una breve descripción de los distintos componentes que encontramos en nuestro proyecto.

2.4.1 HDFS

HDFS es principalmente un sistema de almacenamiento distribuido utilizado por las aplicaciones de Hadoop. Se trata un sistema de ficheros distribuido diseñado para ejecutarse sobre Hardware de simples prestaciones. La diferencia que tiene con otros sistemas de ficheros distribuidos son significantes:

- Tolerancia ante fallos.
- Bajo coste de recursos hardware
- Provee de un amplio caudal de acceso a los datos para las aplicaciones, lo que es útil para aplicaciones que tiene un gran conjunto de datos.

HDFS tiene una arquitectura maestro/esclavo consistente en dos elementos:

- **NameNode :** En un clúster HDFS sólo hay un único NameNode, el maestro que gestiona el espacio de nombres del sistema de ficheros y regula su acceso a los clientes.
- **DataNode :** Hay un número de DataNodes, normalmente uno por nodo en el clúster, el cual gestiona el almacenamiento asignado a los nodos que son ejecutados.

Ambos componentes son piezas de software diseñadas para ejecutarse con comodidad en una máquina.

2.4.2 YARN

Apache YARN es un sistema de gestión de recursos de los clústers de Hadoop. Tiene principalmente dos componentes:

- **ResourceManager :** Se trata de la máxima autoridad que arbitra los recursos entre todas las aplicaciones en el sistema. A su vez tiene dos componentes principales:
    - **Scheduler :** Es el responsable de asignar los recursos de las diversas aplicaciones que están en ejecución dentro de las restricciones de capacidad, colas, etc…
    - **ApplicationsManager :** Es el responsable de aceptar los trabajos, la negociación de los contenedores para ejecutar una aplicación específica y proporcionar el servicio de reseteo en caso de falla.

- **NodeManager :** Se trata del agente responsable de los contenedores, monitorizar los recursos en uso

(CPU, memoria, disco, red) y mantener informado al ResourceManager.

2.4.3 Samza

Dentro de las aplicaciones que puede ejecutar Hadoop encontramos Samza, que es básicamente un framework para el procesamiento de flujos. A continuación describiremos de forma breve los componentes de Samza, dado que es en ella donde se alojará la aplicación relacionada con MOA.

- **Stream :** El Stream (o flujo) está compuesto por una cantidad de mensajes inmutables de un tipo o categoría similar. Samza soporta la adición se sistemas que implementan la abstracción de flujos. En nuestro proyecto el sistema que nos proporciona los flujos es Kafka los cuales son los denominados topics.

- **Job :** En Samza tenemos el concepto de Jobs (o trabajos) estos Jobs permiten la ejecución de aplicaciones que transforman un conjunto de flujos de entrada pudiendo ofrecer posteriormente un nuevo flujo de salida con la transformación efectuada.

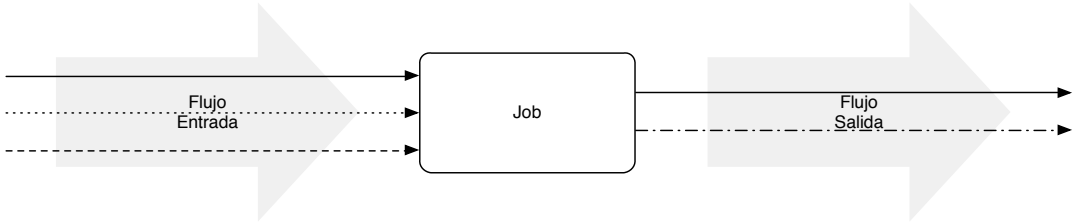

Figura 2-9. Diagrama de un trabajo de Samza procesando flujos

- **Tareas :** Las tareas son la unidad de paralización del trabajo al igual que la partición lo es del flujo. Un trabajo puede estar compuesto por múltiple tareas en función del número de particiones en los que esté dividido el flujo procesando los mensajes que llegan de forma secuencial, permitiendo que cada tarea opere de forma independiente.

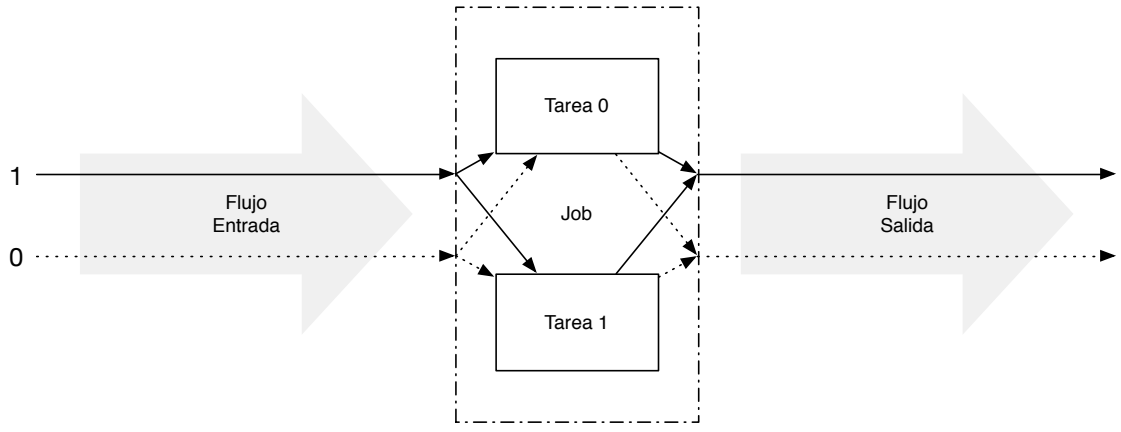

Figura 2-10. Anatomía de un trabajo de Samza

- **Container :** El container es la unidad de paralelización física y se corresponde con la asignación de

los recursos de una computadora (como la RAM o la CPU). Un contenedor puede ejecutar una o varias tareas. Aunque las tareas estén determinadas por el número de particiones de un flujo de entrada, no ocurre lo mismo con los containers, ya que estos están determinados por el usuario, así como el uso de los recursos.

Podemos sintetizar todo lo visto hasta ahora en la siguiente imagen.

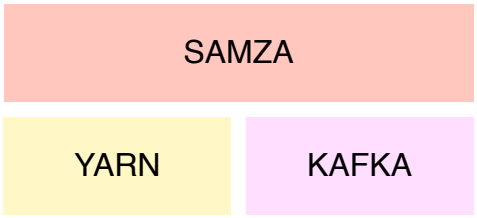

Figura 2-11. Arquitectura final sobre la que funcionará Samza

En ella podemos observar los componentes que se emplearán en el proyecto, observamos que Samza va sobre YARN y Kafka, es lógico pues dado que YARN gestiona los recursos tiene que asignarlos a Samza para que éste pueda ejecutar sus tareas. Del mismo modo Kafka es el cliente utilizado para alimentar a Samza en los flujos de entradas.

3 {M}ASSIVE {O}NLINE {A}NALYSIS

Esto es una cita al principio de un capítulo.

- El autor de la cita -

MOA es un entorno software para implementar algoritmos y llevar a cabo experimentos para el aprendizaje continuo de flujo de datos evolutivos. MOA incluye una colección de métodos incrementales y no incrementales así como herramientas para la evaluación de los resultados, incluyendo una cantidad de algoritmos orientados a Machine Learning (clasificación, regresión, clusterización y detección de anomalías). MOA está relacionado con el entorno de análisis del conocimiento WEKA que tiene una interacción bidireccional con MOA aprovechando para ello algunas de sus características.

Una gran ventaja es que tanto MOA como WEKA están escrito en Java, lo que permite portar las aplicaciones a otros sistemas y por supuesto el fuerte soporte de librerías que están en continuo desarrollo. En este capítulo haremos un análisis sobre las características y el funcionamiento de MOA.

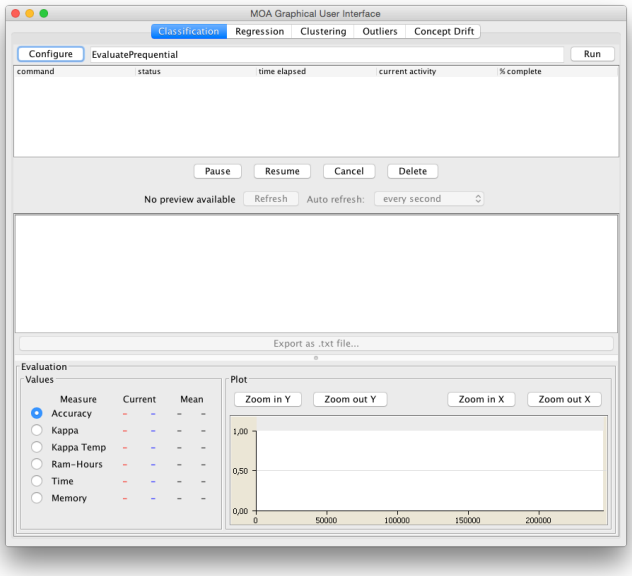

Figura 3-1. Interfaz gráfica de MOA

3.1 Aspectos generales de MOA

MOA es concebido con los problemas de clasificación, quizás la tarea más comúnmente investigada dentro de la rama de machine learning. El objetivo de la clasificación es producir un modelo que pueda predecir la clase a la que pertenece las muestras no etiquetadas, mediante el entrenamiento de muestras cuyas etiquetas, o clases, son suministradas. Para simplificar el problema que se está abordando se hará unas suposiciones sobre el escenario de aprendizaje:

1. Se supone que los datos tienen una cantidad pequeña y fija de columnas, o atributos/características. Cientos a lo sumo.
2. El número de filas, o muestras, es muy elevado, millones de muestras en la escala más pequeña.
3. Los datos pertenecen a un limitado conjunto de clases, normalmente menos de diez.
4. La cantidad de memoria disponible para el algoritmo de aprendizaje depende de la aplicación. La cantidad de datos a tratar se considerará mayor que la cantidad de memoria disponible.
5. Debe haber un pequeño límite superior de tiempo permitido para entrenar o clasificar una muestra.
6. Asumimos que los flujos pueden ser estacionarios o evolutivos.

En un entorno de flujo de datos existen diferentes requisitos, los más significantes son los siguientes:

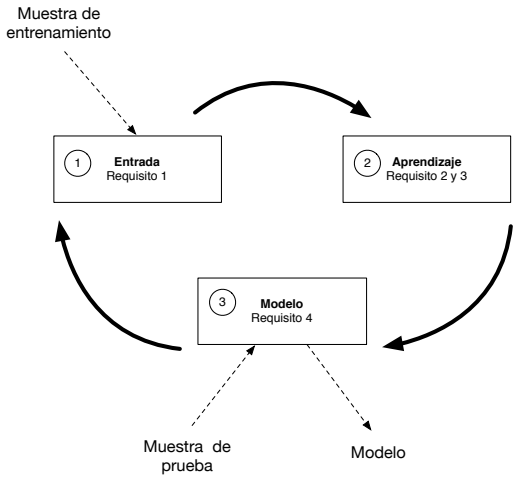

Figura 3-2. Representación del ciclo de aprendizaje y clasificación de MOA

- **Requisito 1**: Procesar una muestra en el instante e inspeccionarla una única vez (al menos)
- **Requisito 2**: Utilizar una cantidad limitada de memoria
- **Requisito 3**: Trabajar en un periodo de tiempo limitado
- **Requisito 4**: Estar preparado para predecir en cualquier instante

En la figura anterior podemos ver el uso común de un algoritmo de clasificación de flujo de datos y cómo los requisitos están fijosdentro de un bucle:

1. El algoritmo obtiene la siguiente muestra del flujo (Requisito 1).
2. El algoritmoprocesa la muestra, actualiza su estructura de datos. Esto lo hace sin exceder los

límites de memoria (Requisito 2) y tan rápido como es posible (Requisito 3).

3. El algoritmo está listo para aceptar la próxima muesra. En una petición puede predecir la clase de muestras ajenas (Requisito 4).

Como mencionamos antes, MOA tiene algoritmos incrementales y no incrementales. La principal diferencia es que los algoritmos incrementales son aquellos en los que el tratamiento de la instancia se hace de forma aislada y secuencial, esto es, de una en una. El aprendizaje incremental tiende a descartar o reajustar continuamente el modelo generado.

Los algoritmos no incrementales no pueden aplicarse en entornos evolutivos donde los datos van cambiando y obteniendo con el tiempo, es el conocido como batch o procesamiento por lotes. Los modelos construidos por estos algoritmos tienden a ser corregidos y ampliados.

3.2 Componentes de MOA

En esta sección hablaremos de los componentes básicos para llegar a comprender el funcionamiento de MOA, estos componentes son clave dentro de la rama de machine learning.

3.2.1 Definiciones

Antes de proceder con la definición de los componentes de MOA es necesario dar una breve definición de algunos términos que usaremos a lo largo de este capítulo.

3.2.1.1 Concepto

Basicamente hay cuatro tipos de estilos diferentes de aprendizaje en las aplicaciones de data mining. En la clasificación, el esquema aprendido es presentado como un conjunto de muestras clasificadas desde las cuales se espera aprender una forma de clasificar las muestras no visualizadas. En la asociación, se busca cualquier asociación entre las características, no sólo aquellas para predecir un valor de una clase particular. En clustering, se busca el agrupamiento de muestras que pertenecen a un mismo tipo. En la predicción numércia, el resultado que se predice no es una clase discreta, sino una cantidad numérica. Independientemente del tipo de aprendizaje involucrado, llamaremos a esa cosa a ser aprendida el concepto y a la salida producida por el aprendizaje la descripción del concepto.

3.2.1.2 Muestra

La entrada a un máquina aprendiz suele ser un conjunto de instancias. Estas instancias son las cosas que pueden ser clasificadas, asociadas o clusterizadas. Aunque hasta ahora las hemos llamado muestras, de aquí en adelante usaremos un término más específico denominado instancia para referinos a la entrada. En un escenario normal, cada instancia es una individual e independiente muestra de el concepto a set aprendido. Las instancias están caracterizadas por los valores de un conjunto de atributos predeterminados. Las instancias pueden ser agrupadas de manera que forman un dataset.

3.2.1.3 Atributo

Cada instancia proporcionada a la entrada de una máquina aprendiz es caracterizada por unos valores fijos, un conjunto de características o atributos predefinidos. El uso de un conjunto fijo de atributos impone otra restricción al tipo de problemas que generalmente se considera en la minería de datos. Así por ejemplo las

instancias fueran vehículos de transportes, el número de ruedas es una característica que se aplica a la mayoría de los vehículos, pero no a las embarcaciones, por ejemplo.

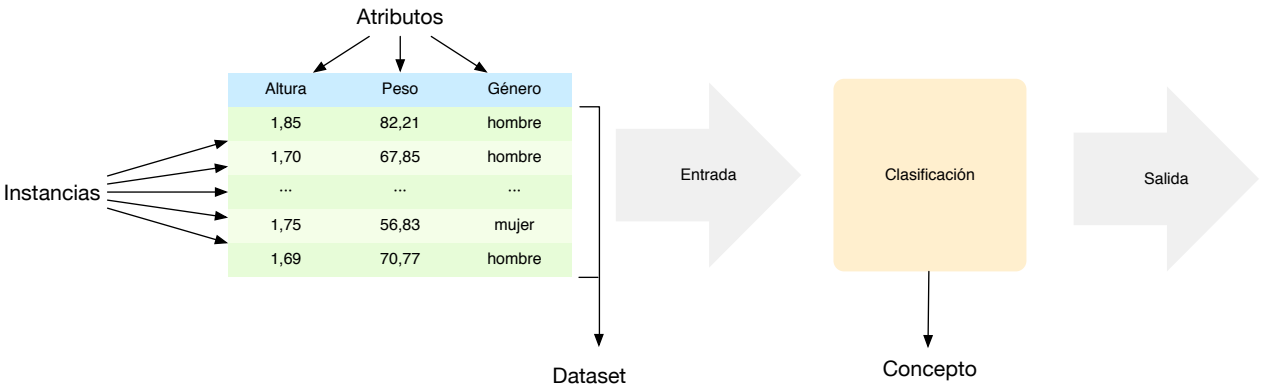

Figura 3-3. Esquema de aprendizaje aplicando un algoritmo de clasificación

En la figura anterior puede verse de forma gráfica cada una de las definiciones. Supongamos que tenemos un algoritmo de clasificación, el hecho de clasificar y distinguir, por ejemplo, qué personas tienen sobrepeso y cuáles no es lo que denominamos concepto. Para ello necesitamos un conjunto de atributos que son las columnas de nuestro dataset (altura, peso y género). Y por último las instancias que son los valores fijos que tienen los atributos y que forman las filas de nuestra tabla. Cada una de estas instancias será la entrada al algoritmo que tras su aplicación mostrará una serie de resultados a su salida.

3.2.2 Instancia

La unidad de aprendizaje de MOA, es la instancia, una instancia no es más que una muestra del conjunto de instancias que representa el entrenamiento de nuestro aprendiz. Una instancia, por tanto, se utiliza como fuente de análisis en el aprendizaje realizado por el detector de anomalías que abarcaremos más adelante. Cada instancia está formada por un vector de valores en coma flotante de doble precisión. Estos valores están contenidos dentro de los atributos, cuyos tipos se resumen en la siguiente tabla:

Tabla 3–1 Tipos de atributos y su representación en MOA

Tipo de atributo	Representación
Numérico	Número en coma flotante
Nominal	Conjunto fijo de valores nominales
Cadena	Conjunto de valores nominales dinámicamente extensible.
Fecha	Fecha, internamente se representa como un número en coma flotante almacenado como los milisegundos desde el 1 de Enero de 1970 00:00:00 GMT. La cadena de representación de las fechas deben de seguir la ISO-8601
Relacional	Puede contener otros atributos. Usado para representar datos multivaluados.

A continuación mostramos como instanciar en Java los atributos más comunes de MOA:

```
// Atributo numérico
Attribute altura = new Attribute(“altura”)

// Atributo nominal
List tipos_genero = new ArrayList(2);
tipos_genero.add(“hombre”)
tipos_genero.add(“mujero”)
Attribute genero = new Attribute(“genero”);

// Atributo de cadena
Attributo nombre = new Attribute(“nombre”, (ArrayList) null);
```

Dentro de una instancia tenemos un valor denominado clase o etiqueta, que hace referencia a los distintos tipos en los que pueden clasificarse las muestras de entrenamiento, en la siguiente figura podemos ver la representación de una instancia de MOA, en ella podemos apreciar los atributos y la clase:

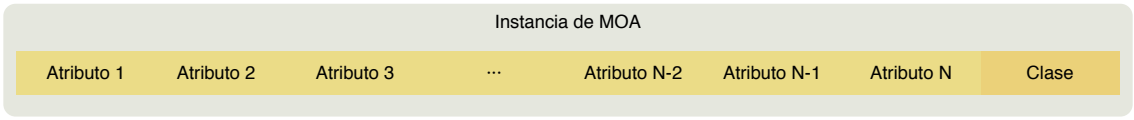

Figura 3-4. Ejemplo genérico de una instancia en MOA

Así podemos crear, por ejemplo, una instancia con la siguiente estructura:

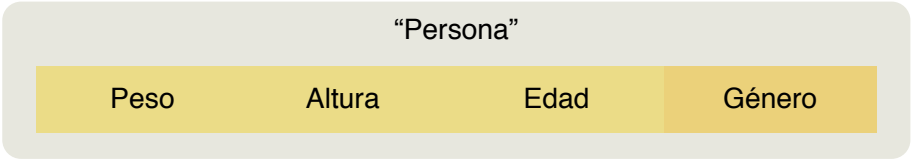

Figura 3-5. Ejemplo de la instancia Persona

En ella podemos observar tres atributos (peso, altura y edad) y una clasificación basada en el género (masculino o femenino). No obstante dentro de las instancias que puede manipular MOA podemos distinguir dos tipos de instancias:

- **DenseInstance :** En la que todos los valores (numéricos, fecha, nominal, cadena y relacional) están almacenados internamente como números en coma flotante.
- **SparseInstance :** En la que sólo requiere el almacenamiento de aquellos atributos cuyos valores no son cero. Tiene como objetivo reducir la ocupación en memoria.

Todas estas estructuras e implementaciones se han tomado prestadas de WEKA, permitiendo una comunicación bidireccional con MOA.

3.3 Detector de anomalías

La detección de anomalías es una tarea importante dentro de la minería de datos cuyo objetivo es descubrir aquellos elementos que muestran una desviación significante del comportamiento esperado, tales elementos son denominados outliers (valores atípicos), mientras que aquellos que estén dentro del comportamiento normal se considerarán inliers.

Uno de los métodos más extendidos para determinar si un elemento es outlier es basarse en el número de elementos vecinos dentro de una distancia R frente a un umbral k. Esos outliers son denominados como valores anómalos basados en distancia.

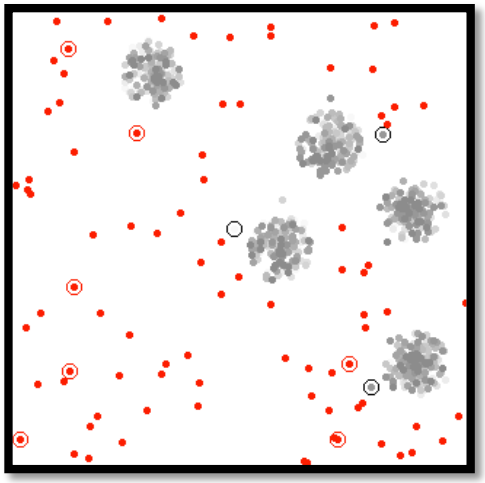

Figura 3-6. Detector de anomalías en funcionamiento

En la imagen anterior podemos ver una instantánea de un detector de anomalías en funcionamiento, pueden observarse que aquellos puntos de impacto en color rojo presentan una característica anormal, por lo que son clasificados como outliers. Podemos distinguir otros puntos en gris que indican un comportamiento dentro de la normalidad, estos puntos de impactos son los denominados inliers.

3.3.1 Motivación

La detección de anomalías es una aplicación muy común dentro de la rama de machine learning. Normalmente los detectores de anomalías se basan en la medición estadística de las características de aquellas muestras que se quieren procesar.

Para ello se construye un modelo a partir de los datos de entrenamiento y se plantea la probabilidad de que una muestra X sea considerada normal, de forma que si dicha muestra se encuentra fuera de cierto umbral se considerará outlier y si se encuentra dentro del umbral se considerará inlier. No obstante podemos encontrar casos límite en lo que una muestra se pueda considerar de ambas formas, pues no se puede determinar a priori su clasificación.

Algunos de los casos usos de la detección de anomalías son los siguientes:

- **Detección de fraude** : Los usuarios tienen una actividad asociadas a ellos, tales como: El tiempo que permanecen online, su localización a la hora de registrarse en el sistema o los gastos efectuados. Usando estos datos podemos construir un modelo de la actividad normal de un usuario e identificar usuarios sospechosos permitiendo tomar medidas al respecto.
- **Manufacturación**: La fabricación de motores de aeroplanos es una maravilla de la ingeniería en cuanto a eficiencia, potencia y consumo. La detección de comportamientos inusuales en algunas de esas características puede suponer un riesto a la hora de su utilización, es por ello que el uso de los detectores de anomalías puede ayudar a detectar tales comportamientos y aplicar un correctivo ante esos casos.
- **Monitorizando data center**: Si tenemos muchas máquinas en un clúster tal vez nos gustaría monitorizar la memoria que utilizan, el número de accesos al disco por segundo, la carga de CPU o, de forma más compleja, la carga de CPU según el tráfico de red. Si aplicamos un detector de anomalías y observamos un comportamiento anómalo podríamos estar ante un posible fallo y llegar a la conclusión de un reemplazo en caso de tener que hacerlo.

Como puede verse la cantidad de aplicaciones son variadas e interesantes. MOA implementa una serie de algoritmos para la detección de anomalías basándose en la distancia más que en la probabilidad. En el siguiente apartado definiremos el funcionamiento de dichos detectores así como una característica interesante de éstos.

3.3.2 Funcionamiento

Los detectores de anomalías basados en distancia tienen un funcionamiento que sigue la siguiente definición: Un objeto x es marcado como outlier si hay menos de k objetos localizados a una distancia máxima R de x.

En la imagen inferior podemos ver cuatro puntos de impactos a los cuales se les atribuye un radio R y un umbral $k = 3$. Los puntos $P1$, $P2$ y $P3$ son considerado inliers, pues dentro de su radio encuentran 3 vecinos próximos, cumpliendo con el umbral. Sin embargo el punto $P4$ es considerado un outlier, ya que dentro de su radio sólo se encuentra un vecino y por lo tanto no cumple con el criterio del umbral.

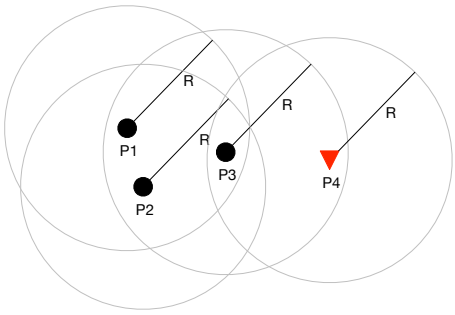

Figura 3-7. Ejemplos de resultados obtenidos para un detector con umbral k = 3.

Este proyecto está enfocado en la búsqueda anomalías en flujos de datos, donde continuamente están llegando nuevos objetos mientras que los más antiguos expiran. Para poder controlar estos objetos el detector posee una

ventana deslizante basada en contadores, donde cada vez que un nuevo objeto llega el más antiguo expira manteniendo así un conjunto de objetos activos de forma constante.

Esto es un desafío frente a datos estáticos, principalmente por la naturaleza de los objetos. Un objeto puede cambiar su estado a lo largo de su vida. Por ejemplo un objeto x que fue un outlier en t_1 podría perder esta propiedad en t_2 y convertirse en un inlier. Evidentemente podría haber objetos que pueden tener ambos estados (inlier y outlier) como se mencionó anteriormente.

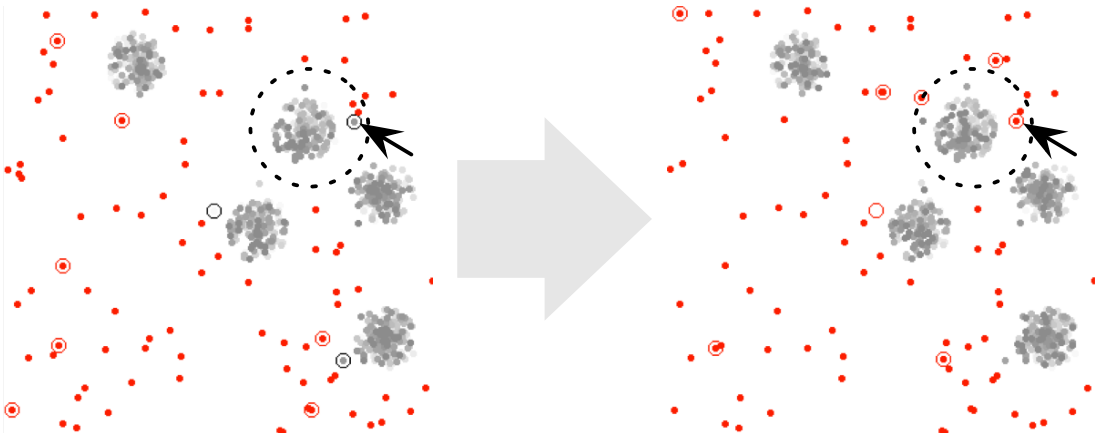

Figura 3-8. Ejemplo de un punto pasando de Inlier a Outlier tras 100 instancias

En la imange superior puede observarse el caso de dos instantáneas en t_1 y t_2, en ellas señalamos uno de los clústers en los que un punto de impacto pasa de ser un inlier a ser un outlier.

Con todo lo visto por ahora podemos definir un detector basado en distancia con sólo tres parámetros: ventana, umbral de vecinos y radio.

De todos estos parámetros el que juega un papel más importante es la ventana, ya que en esta se mantendrá en memoria y servirá para cotejar los nuevos valores recibidos decidiendo si dicho valor es un outlier o no. Hay que tener en cuenta que la ventana es un rango de consultas que hay que realizar por cada nuevo objeto recibido, de manera que si tenemos una ventana muy grande podemos llegar a tener consultas con una inversión de tiempo muy elevada, lo que rompería el concepto de real-time, sin embargo si la hacemos muy pequeñas podemos entrar en la posibilidad de encontrar demasiados outliers.

3.3.3 Algoritmos

MOA ofrece una pequeña colección de algoritmos para la detección de anomalías, daremos una breve explicación sobre ellos:

STORM : Por cada nuevo objeto p que llega se comprueba que el número de vecinos que se ha analizado anteriormente estén dentro de su radio R y que cumpla con el umbral k. Además, por cada nuevo objeto que llega, una serie de consultas con radio R es llevada a cabo para determina el nuevo vecino, el cual es añadido a su lista de vecinos analizados. En cualquier instante de tiempo, la aproximación adoptada por STORM decide si un objeto p es anómalo computando el objeto en la lista de vecinos analizados que no hayan expirado aún.

El coste computacional para un objeto es de $O(\log k)$, lo que significa que el coste para el total de los objetos es de $O(n \log k)$.

Abstract–C : Esta aproximación reduce el coste a $O(n)$, ya que mantiene continuamente el número de vecinos de un objeto para toda la ventana hasta que expire. Por eso, la aproximación de Abstract–C tiene el peor caso en cuento al requerimiento de espacio, es decir $O(n * windowsize)$. En el peor de los casos, el espacio necesario puede llegar a ser igual a $O(n^2)$. Además, cada uno de los contadores de la ventana podría ser actualizado varias veces antes de que llegue a considerarse obsoleto. Sin embargo, tiene la capacidad de responder múltiples consultas con múltiples valores de k, y puede trabar tanto con ventanas de tiempo como ventanas de conteo.

COD : Este algoritmo en lugar de comprobar cada objeto de forma continua, computa el siguiente instante de tiempo futuro en el que un objeto puede ser clasificado como outlier e inspecciona un único objeto en cada instante de tiempo. Además, utiliza una estructura de cola para almacenar los objetos. Tal estructura soporta la detección de los objetos que podría llegar a a ser outliers con un coste de $O(1)$. La principal diferencia con STORM es que el número de objetos que necesitan ser examinados en cada pasada de la ventana es significativamente menor. Comparado con Abstract–C, es más rápido y necesita menos espacio.

MCOD : Construido sobre COD y empleando las mismas colas de evento, su característica distintiva es que mitiga la necesidad de evluar el rango de consultas para cada nuevo objeto con respecto a todos los demás objetos activos. Su solución está basada en el concepto de micro-clúster evolutivos que corresponde con las regiones que contiene inliers exclusivamente. Posteriormente el rango de consultas son computadas por cada nuevo objeto con respecto al micro-clúster.

De forma resumida, la aproximación con STORM tiene requisitos de memoria aceptable ($O(nk)$), despreciable tiempo para actualizar la información de cada objeto existente debido a la llegada de nuevos objetos y la expiración de los antiguos (con un coste de $O(1)$ por cada objeto), pero un significante tiempo de producción de outliers ($O(n \log k)$). Por el otro lado, Abstract-C tiene altos requisitos de memoria, altos requisitos de ejecución para actualizar la infomación existente debido a los cambios en la población de la ventana (ambos con un coste de $O(n * windowsize)$ por cada nuevo objeto) y un bajo coste de tiempo de producir los outliers $O(n)$. COD y MCOD tienen un coste de $O(n)$ en las necesidades de espacio y son más rápidos de STORM y Abstract-C.

Así podemos determinar que debido a que necesitamos procesar los eventos de seguridad en el menor tiempo posible es necesario adoptar un algoritmo que contribuya a dicho fin, este algoritmo será MCOD pues posee todas las ventajas de COD y las características de los detectores basados en distancia. Además los requisitos de espacios son menores que STORM y Abstract–C.

3.3.4 El problema de la pérdida de información

Los algoritmos que hemos expuesto de MOA ofrecen detectores de anomalías con diversas características, sin embargo, el corpotamiento de MOA frente a estos detectores se ve afectado por el uso de las instancias de WEKA mencionadas anteriormente. Recordemos que dichas instancias son vectores de valores en coma flotante de doble precisión, por lo tanto no son capaces de interpretar valores literales y estos son asignados como valores enteros dentro de la estrucutra de valores.

Para hacernos una idea de lo que esto representa vamos a realizar un breve ejemplo. Supongamos que tenemos nuestro dataset con sus dos atributos y una clasificación que son peso, altura y género respectivamente. Las instancias de MOA dentro del dataset tendrán por lo tanto la siguiente estructura:

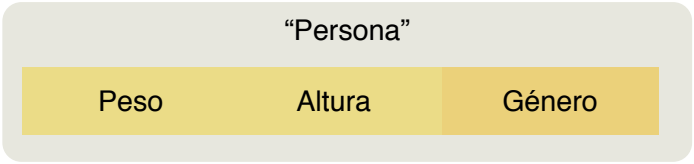

Figura 3-9. Instancia persona con dos atributos y una clase

Supongamos que el detector de anomalías está pensado para detectar casos de sobrepeso en función de la altura y el peso, según sea hombre o mujer, para ello vamos a analizar el comportamiento de los detectores ante los atributos literales y numéricos. Para ello elaboraremos unas gráficas que representarán un instante de tiempo del detector pero con la característica de poder ver los resultados interpretable por el lector.

En la figura inferior podemos ver cómo interpreta el detector las instancias basadas en valores numéricos, si observamos la información que se obtiene acerca del dataset, puede verse claramente el valor computado por el detector ante cada una de las instancias, distinguiendo los datos y siendo comprensibles para nosotros. Esto es posible dado que tanto la altura como el peso son valores numéricos y pueden ser representados como valores en coma flotante.

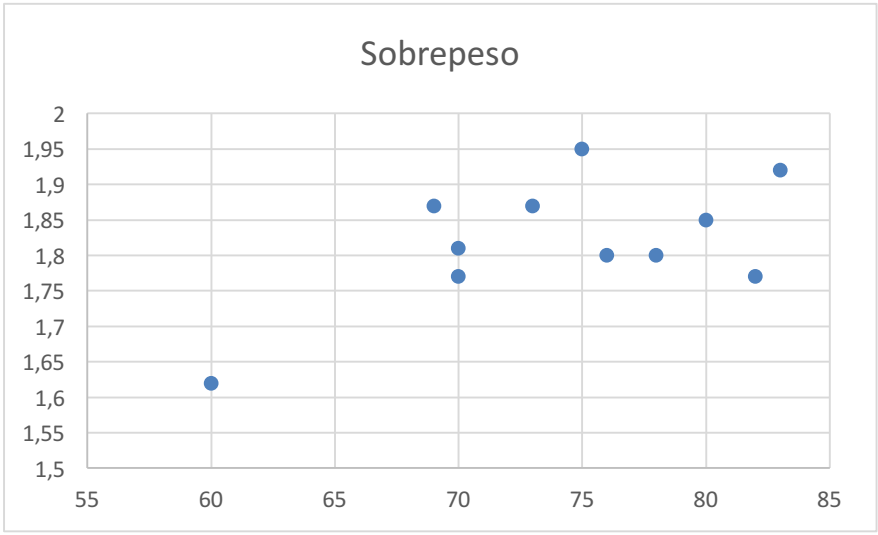

Figura 3-10. Visión del detector con atributos numéricos

Si ahora decidimos utilizar los valores de forma literal, el detector lo interpretaría como un número entero en coma flotante, esto es debido a que en la computadora una cadena te texto como es “1,80” o “76,21” no puede ser entendida como un valor en coma flotante. Su visión se basará en asignar un índice a cada cadena e incrementarlo por cada cadena nueva que aprenda, de manera que se pierde información y por lo tanto podemos obtener una vista del detector muy diferente a la anterior, viéndose de la siguiente forma:

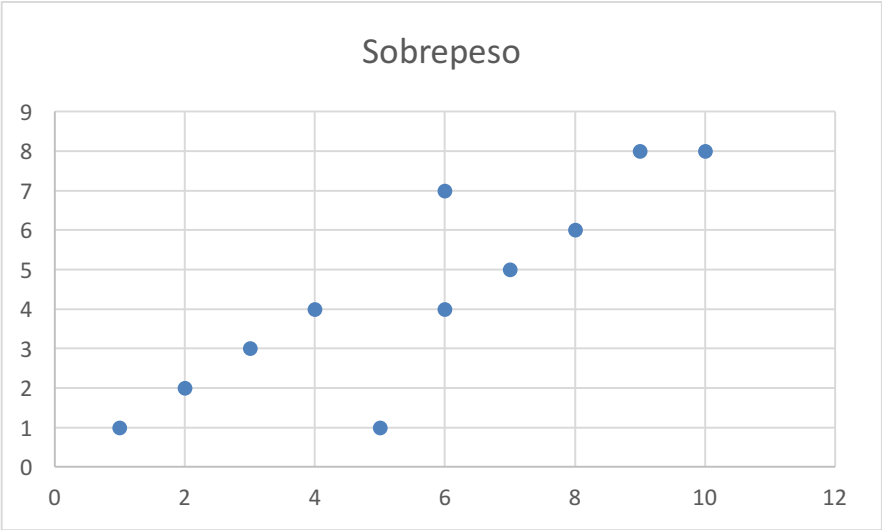

Figura 3-11. Visión del detector con atributos literales

Podemos ver que se ha perdido información, no podemos interpretar los datos de forma adecuada ni existe una clusterización de estos, simplemente una distribución entera dentro de la ventana deslizante del detector.

Estos factores hay que tenerlos en cuenta a la hora de monitorizar los atributos, pues de ello depende el tiempo de procesado del detector. Un claro ejemplo de esta importancia es su aplicación con direcciones IPv4.

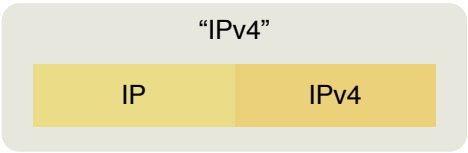

Figura 3-12. Instancia IPv4 de una dimensión

Si suponemos un rango de IPv4 de por ejemplo 10.0.150.1 a 10.2.150.1 y la interpretamos de forma literal podemos comprobar que tenemos más de 16646144 hosts, esto se puede traducir como una cantidad de procesamiento muy elevada, ya que para encontrar outliers necesitaríamos una ventana de un tamaño considerable. Al tratar las direcciones IPv4 como literales estamos definiéndolas de la siguiente manera: “10.0.150.1”, “10.0.150.2”, …, “10.2.150.1”. Lo que se traduce a más de 16646144 valores enteros posibles, es decir: 1, 2, …, 16646144, etc…

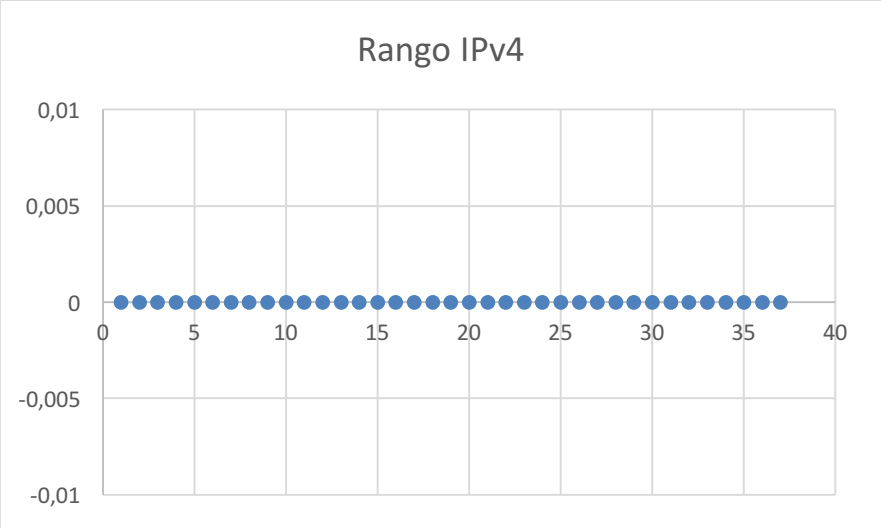

Figura 3-13. Visión del detector con atributos literales en un rango IPv4

En la imagen superior podemos ver la visión del detector, los datos sobre direcciones IPv4 se han perdido y además el coste computación cada vez se hace mayor debido a la ventana deslizante del detector. Para solucionar eso debemos de cambiar la dimensión de la instancia. Para el ejemplo anterior si tenemos dos atributos (peso y altura) entonces estamos hablando de una instancia de dos dimensiones. Por lo tanto podemos decir de forma genérica que la dimensión vendrá dada por el número de atributos que tiene una instancia sin contar con su atributo de clase.

Como bien sabemos, una dirección IPv4 está formada por 4 octetos. Estos octetos los podemos separar y de esa forma crear una instancia con 4 atributos numéricos de la forma A.B.C.D. Esto significa que la instancia tiene 4 dimensiones como puede observarse en la siguiente figura.

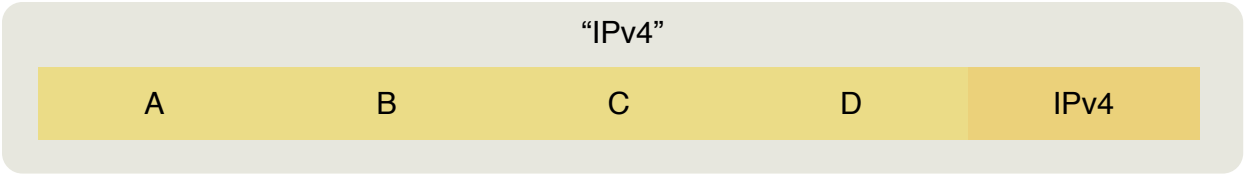

Figura 3-14. Instancia IPv4 de cuatro dimensiones

La representación de 4 dimensiones es algo compleja, pero podemos optar por una visualización aproximada proyectando uno de los bloques sobre las otras tres dimensiones. Si observamos la siguiente imagen:

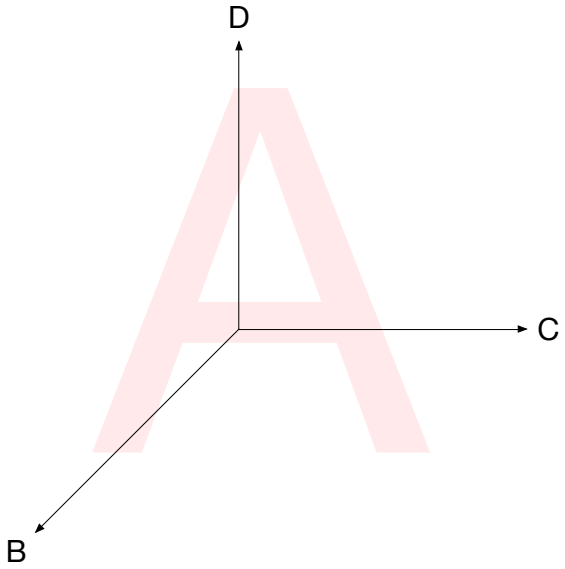

Figura 3-15. Visión del detector con atributos numéricos en un rango IPv4

Podemos observar que para los bloques B, C y D tenemos 3 dimensiones definidas y el bloque A será una dimensión más que servirá como un espacio dentro del cual encontraremos los bloques B, C y D. De esta forma podemos mejorar el tiempo de computación del detector en la ventana deslizante y el detector de anomalías basaría sus detecciones en los cambios dentro de las dimensiones en lugar de observar cambios dentro de los valores numéricos.

Este pequeño análisis que hemos realizado es característico para las direcciones IP y no es la única forma, ya que puede hacerse de muchas otras más. Hay que tener en cuenta que la configuración de los parámetros del detector deben ser específicos para los atributos o grupo de atribuos y en sí es un caso de estudio interesante.

4 INTEGRACIÓN DE MOA EN SAMZA

Ya tenemos una base lo suficientemente estable como para empezar a construir en ella. En este capítulo entraremos por completo en la integración de MOA en una tarea de Samza mostrando los resultados obtenidos. Para ello pondremos en práctica todos y cada uno de los conceptos redactados a lo largo de esta memoria.

Mostraremos de forma detallada la implementación de una tarea en Samza así como el uso del algoritmo MCOD que utilizaremos para la detección de anomalías. También mostraremos el uso del productor de eventos sintéticos y pasaremos a probar, analizar y estudiar la viabilidad de MOA para un entorno Big Data.

4.1 Implementación de una tarea Samza

La elaboración de una tarea para Samza es tan sencillo como la implementación de las interfaces que ofrece para dicho fin.

4.1.1 Procesado

Para el procesado de los flujos de entrada es necesario la implementación de la interfaz `StreamTask`. Esta interfaz proporciona el siguiente método:

```
public void process(IncomingMessageEnvelope envelope, MessageCollector collector,
TaskCoordinator coordinator)
```

Por cada mensaje que Samza recibe desde el flujo de entrada, el método `process` es llamado. El objeto `envelope` de la clase `IncomingMessageEnvelope` tiene tres métodos importantes:

- **`Object getMessage()`** : Permite obtener el mensaje recibido del flujo.

- **`Object getKey()`** : Permite obtener la clave del mensaje recibido.
- **`SystemStreamPartition getSystemStreamPartition()`** : Permite averiguar de dónde viene el mensaje que se ha recibido.

Si observamos los métodos `getMessage()` y `getKey()` devuelven un objeto de la clase `Object` al que se le debe aplicar un `cast` al tipo de clase correspondiente. Recordemos que Kafka puede recibir en los topics cualquier tipo de estructura de datos, incluido los binarios.

Como mencionamos antes, el método `getSystemStreamPartition()` devuelve un objeto del tipo `SystemStreamPartition`, que nos dice de dónde viene el mensaje**.** `Esta clase tiene los siguientes métodos:`

- **`String getSystem() :`** Nos devuelve una cadena con el nombre del sistema del cuál proviene el mensaje.
- **`String getStream() :`** Nos devuelve una cadena con el nombre del flujo/topic/cola que está dentro del sistema.
- **`Partition getPartition() :`** Nos devuelve la partición correspondiente del flujo recibido.

Los métodos anteriores devuelven información relacionada con el sistema que emite los flujos que alimentan las tareas. Esta información es obtenida del fichero de configuración, del cual hablaremos más adelante.

Una interfaz de interés para nosotros es `MessageCollector`. Dicha interfaz contiene el siguiente método:

- **`void send(OutgoingMessageEnvelope envelope) :`** Método que nos permite enviar un mensaje a un sistema.

Para poder emitir un mensaje desde el colector es necesario crear una objeto del tipo `OutgoingMessageEnvelope` y pasárselo. Como mínimo hay que establecer el mensaje a enviar, y el sistema y flujo destino. Opcionalmente también se puede espcificar la clave de partición y otros parámetros los cuáles puede revisar en la documentación de Samza.

El objeto que implementa la interfaz `TaskCoordinator`, posee otros métodos que no utilizaremos en este proyecto y que incluye, entre otras cosas, la capacidad de apagar una tarea (o varias) dentro del container.

4.1.2 Inicialización

Cuando un contenedor comienza, crea una instancia de la tarea que se haya implementado por cada partición del flujo que recibe. Samza ofrece la posibilidad de programar acciones previas al inicio de la tarea.

```
public void init(Config config, TaskContext context)
```

Cada vez que una tarea de samza sea inicializada se llamará el método `init` el cual recibe dos objetos de la clase `Config` y `TaskContext`. Ambas proporciona funcionalidades con respecto al archivo de configuración de

Samza y los flujos. Nos centraremos sobretodo en el archivo de configuración, al cuál entraremos en la siguiente sección.

4.2 Configuración de una tarea Samza

Todos los trabajos de Samza tienen un fichero de configuración asociado. Una configuración básica está compuesta por la siguiente estructura como mínimo:

```
# Job
job.factory.class=org.apache.samza.job.local.ThreadJobFactory
job.name=hello-world

# Task
task.class=samza.task.example.MyJavaStreamerTask
task.inputs=example-system.example-stream

# Serializers
serializers.registry.json.class=org.apache.samza.serializers.JsonSerdeFactory
serializers.registry.string.class=org.apache.samza.serializers.StringSerdeFactory

# Systems
systems.example-system.samza.factory=samza.stream.example.ExampleConsumerFactory
systems.example-system.samza.key.serde=string

systems.example-system.samza.msg.serde=json
```

En la sección del trabajo (Job) se define el nombre del trabajo, y si utiliza la clase de `YarnJobFactory` o `ProcessJobFactory/ThreadJobFactory`. Estas factorías son necesarias para poder comenzar el trabajo dentro del container. Las características de cada una son:

- **`ThreadJobFactory :`** Ejecuta el trabajo en una máquina local usando hilos. Esta opción está pensada para entornos de desarrollo, no para los de producción.
- **`ProcessJobFactory :`** Ejecuta el trabajo en una máquina local como un subproceso. Esta opción está pensada para entornos de desarrollo, no para los de producción.
- **`YarnJobFactory :`** Ejecuta el trabajo sobre YARN. Pensado para entornos de producción.

En la sección de la tarea (Task) es donde se especifica el nombre de la clase de tu flujo de tarea. Es también donde se define los flujos de entrada que alimentarán la tarea.

En la sección de serializadores (Serializers) es donde se especifica la clase que se utilizará para la serialización y deserialización de los datos, en este ejemplo encontramos dos tipos, **`JsonSerdeFactory`** y **`StringSerdeFactory`**, cada uno encargado de serializar y deserializar JSON y String respectivamente.

En la parte de sistemas (Systems) se define qué sistemas puede leer la tarea implementada con los tipos de serdes usados para enviar claves y mensajes a otros sistemas. Nososotros utilizaremos Kafka como sistema de alimentación de la tarea, aunque pueden definirse otros de forma personalizada.

Este ejemplo de fichero de configuración nos ha permitido hacer un breve análisis sobre las necesidades a la hora de ejecutar una tarea en Samza. Nosotros tomaremos ventaja de este fichero pues lo aprovecharemos para definir los parámetros del detector de anomalías así como los atributos y las clases de las instancias que

monitorizará. Para ello se han definido una serie de atributos mostrados en la siguiente tabla:

Tabla 4–1 Listado de atributos para configurar el detector de anomalías

Atributo	Descripción
`outlier.input`	Topics de entradas (flujos) que se monitorizarán. Deben ser valores separados por comas: topic1, topic2, …, topicN
`outlier.review`	Atributo que indica cada cuánto tiempo quiere que se revisen las estadísticas. Número entero mayor o igual a 1.
`outlier.train`	Número de instancias que se utilizarán para entrenar al detector antes de pasar a la detección de anomalías.
`outlier.topic.classes`	Conjunto de atributos que serán monitorizados por el detector, con ellos se definirán las clases de cada uno. Valores separados por comas: attr1, attr2, …, attr3
`outlier.topic.class.name`	Nombre de la clase a la que se definirá el tipo de atributo al que pertenece: string, numeric, IP
`outlier.topic.class.parameters`	Atributo utilizado para configurar los parámentros del detector: ventana, radio y umbral.

Recordemos que las tareas de Samza tienen la posibilidad de implementar un método de inicialización el cual recibe un objeto del tipo `Config` gracias a este objeto obtendremos la configuración de forma automática y simplemente deberemos de obtener los valores, comprobando siempre que se han definidio dichos atributos.

4.3 Sobre los eventos de seguridad

Ya tenemos claro cómo implementar una tarea de Samza, pero tenemos que observar en detalle los eventos de seguridad. Antes de comenzar con la implementación, necesitamos saber cómo son los eventos de seguridad que recibirá la tarea. Un evento de seguridad presenta la siguiente estructura:


```
{"dst_country_code":"US", "engine_id_name":"IANA-L4", "application_id_name":"APP B",
"biflow_direction":"initiator", "dst":"80.82.34.22", "client_rssi_num":-35,"ip_protocol_version":4,
"first_switched":1441538681, "type":"NetFlowv10", "dot11_status":"UNKNOWN", "pkts":321,
"building":"Building A", "l4_proto":5, "src_net_name":"0.0.0.0/0", "flow_end_reason":"idle timeout",
"client_mac_vendor":"SAMSUNG ELECTRO-MECHANICS", "src_country_code":"US", "src_net":"0.0.0.0/0",
"dst_net":"0.0.0.0/0", "floor":"Floor C", "sensor_ip":"90.1.44.3", "direction":"ingress",
"timestamp":1441538681, "wireless_id":"SSID_C", "client_mac":"00:11:22:33:44:58",
"dst_net_name":"0.0.0.0/0", "src":"192.168.1.124", "campus":"Campus D",
"client_latlong":"37.89,26.39", "application_id":"1:9", "src_port":80, "sensor_uuid":7,
"engine_id":5, "bytes":2148, "flow_sampler_id":0, "wireless_station":"00:00:00:00:00:14",
"http":"http://www.google.es", "namespace_uuid":2222, "sensor_name":"sensorB"}
```


Podemos observar que un evento está compuesto por numerosas parejas de clave:valor, pero que en ningún momento contienen vectores internos de éstas.

4.3.1 Producción de eventos

Para poder simular los eventos redBorder ofrece una herramienta que permite generarlos de forma sintética, permitiendo una sin fin de configuraciones. Dicha herramienta utiliza un fichero de configuración para la definir las característica del flujo que producirá los eventos y los topics a los que va dirigo. A continuación mostramos una pequeña tabla en la que se muestran los campos de este fichero de configuración.

Tabla 4–2 Listado de atributos para configurar el productor sintético

Atributo	Descripción
`zk_connect`	Dirección y puerto de ZooKeeper. En él se encontrará el topic a el cual irá dirigido los eventos de seguridad.
`topic`	Nombre del topic al que irá dirigido los eventos de seguridad.
`threads`	Número de hilos para llevar a cabo la ejecución de la herramienta, cada hilo producirá un número de eventos definido por el atributo `rate`.
`rate`	Número de eventos que generará cada hilo.
`fields`	Campos que contendrá el evento de seguridad, puede ser tan extensible como se desee en este atributo.

Gracias a esta herramienta podremos hacer las pruebas necesarias simulando un pequeño entorno Big data en nuestro equipo local y obteniendo las conclusiones sobre la viabilidad de MOA.

4.4 Sobre el detector de anomalías.

Aunque hemos definido el funcionamiento del detector de anomalías, no hemos explicado cómo se instancia y qué métodos ofrece para llevar a cabo su trabajo. La clase para implementar dicho detector se denomina MCOD (coincidiendo con el nombre del algoritmo). Esta clase extiende de **`MyBaseOutlierDetector`** la cual extiende la clase **`AbstractClusterer`**. Como podemos al fin y al cabo aplica el concepto de clúster para llevar a cabo la detección de anomalías y que definimos anteriormente.

Sea cual sea, cada algoritmo de detección de anomalías implementa básicamente los siguientes tres métodos:

- **`void init() :`** Permite inicializar el algoritmo. Es necesario llamarlo para poder utilizar el detector.
- **`void ProcessNewStreamObj(Instance inst) :`** Que añade un nuevo objeto a la ventana actual y, si la ventana está llevan, descarta los objetos más antiguos.
- **`Vector<Outlier> getOutliersResult() :`** Que devuelve el actual conjunto de outliers para su visualización o evaluación.

Podemos observar que la forma de procesar los datos es a través de instancias, tal y como definimos anteriormente.

Además de estos métodos ofrece otro muy útil para conocer la evolución del detector este méodo es **`getStatistics()`** y nos servirá para conocer el número de instancias procesadas, cuáles son outliers, cuales inliers y cuales están en el límite. Además de ello también ofrece información sobre el tiempo de consultas y la memoria ocupada por el algoritmo.

Una características a tener en cuenta de los detectores de anomalías es que una vez que han iniciado su entrenamiento no pueden ser modificados, carece de sentido ya que la ventana deslizante estará actualizada con las instancias procesadas y los resultados no serían válidos.

4.5 Ejecución y puesta en marcha.

Una vez hecho el análisis y compresión de los conceptos, así como la implementación del detector y la lógica de la tarea, vamos a poner en marcha el sistema para comprobar el funcionamiento del detector con estos flujos.

Primero debemos levantar un nodo de ZooKeeper y posteriormente otro de Kafka para que coordine los brokers y registre los topics, posteriormente ejecutaremos YARN que se encargará de gestionar los recursos y asignarlos a Samza una vez se haya subido la tarea. Cuando toda la arquitectura esté preparada comenzaremos a producir los eventos sintéticos de seguridad.

Así pues suponiendo que tenemos levantado los nodos correspondientes pasaremos a subir la tarea a Samza, para ello sólo tenemos que ejecutar el siguiente comando:

```
$ > deploy/samza/bin/run-job.sh –config-
factory=org.apache.samza.config.factories.PropertiesConfigFactory ––config-
path=file://$PWD/deploy/samza/config/datamining.properties
```

Si todos los nodos están activos entonces deberíamos de ver que la tarea se ha subido correctamente, para ello simplemente utilizamos un navegador y vistamos la UI de YARN en el puerto 8088 (por defecto). Primero la tarea se iniciará y se verá con el estado de “aceptada”, posteriormente cuando tenga los recursos asignados pasará al estado de “en ejecución”:

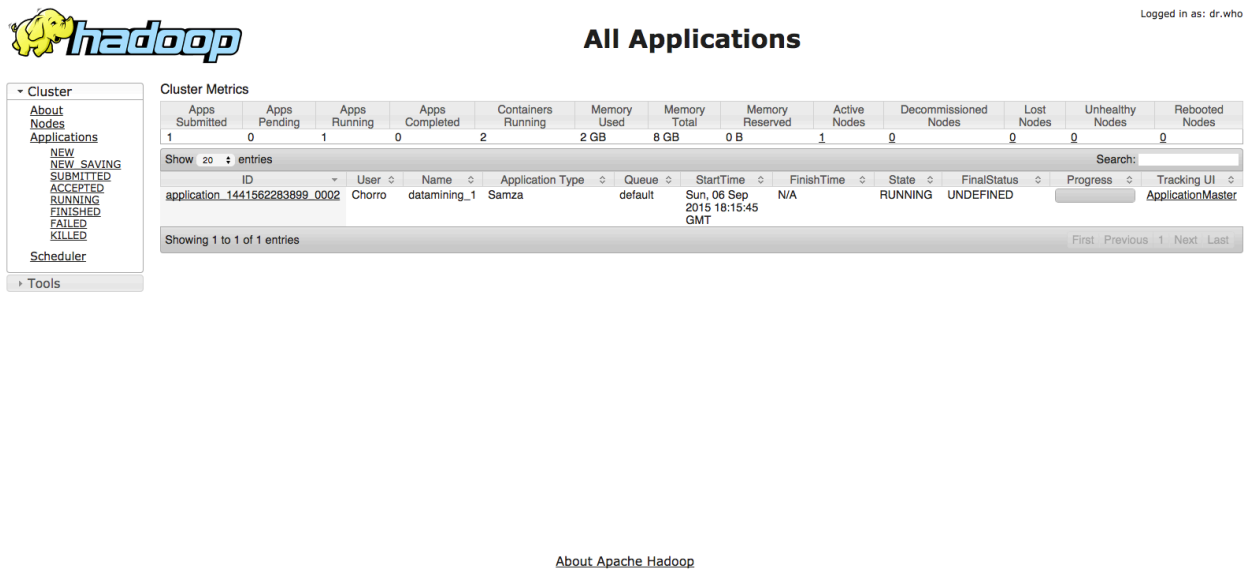

Figura 4-1. Comprobación del estado de ejecución de la tarea en la UI de YARN

Si visualizamos el log que genera la tera de Samza podemos observar que el detector se ha inicializado de forma correcta:

```
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Set review at : 1
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Stream configuration : rb_flow
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added class : src
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added attribute:<Type> : src:IP
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added class attribute with content : [src]
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Created dataset :

Relation Name:  rb_flow_dataset_0
Num Instances:  0
Num Attributes: 5

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 IP_segment_1               Num   0%   0%   0%     0 /  0%     0 /  0%     0
2 IP_segment_2               Num   0%   0%   0%     0 /  0%     0 /  0%     0
3 IP_segment_3               Num   0%   0%   0%     0 /  0%     0 /  0%     0
4 IP_segment_4               Num   0%   0%   0%     0 /  0%     0 /  0%     0
5 class                      Nom   0%   0%   0%     0 /  0%     0 /  0%     0

2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Created new outlier detector
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Setting outlier detector parameters
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Outlier detector prepared with parameters:
        window: 10000
        radius: 10.0
        k: 8
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Adding outlier detector with dataset : rb_flow_dataset_0
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added class : src_country_code
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added attribute:<Type> : src_country_code:string
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Added class attribute with content : [src_country_code]
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Created dataset :

Relation Name:  rb_flow_dataset_1
Num Instances:  0
Num Attributes: 2

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 src_country_code           Str   0%   0%   0%     0 /  0%     0 /  0%     0
2 class                      Nom   0%   0%   0%     0 /  0%     0 /  0%     0

2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Created new outlier detector
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Setting outlier detector parameters
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Outlier detector prepared with parameters:
        window: 100
        radius: 0.1
        k: 20
2015-09-06 20:26:38 AnomalyDetectionTask [INFO] Adding outlier detector with dataset : rb_flow_dataset_1
```

Figura 4-2. Visualización del log generado por la tarea

A continuación pondremos en marcha el productor de eventos sintéticos, para ello ejecutamos el siguiente comando:

```
$ > java -jar synthetic-producer-1.1-SNAPSHOT-selfcontained.jar ../configProducer.yml
```

Podemos observar si se están produciendo eventos si ejecutamos el consumidor de consola de Kafka, también el mismo productor nos informará sobre el número medio de eventos emitidos por segundo. Si observamos nuevamente el log podemos observar que el detector está funcionando de forma adecuada:

```
2015-09-06 20:28:42 AnomalyDetectionTask [INFO] Information of dataset:

Relation Name:  rb_flow_dataset_0
Num Instances:  1
Num Attributes: 5

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 IP_segment_1              Num    0% 100%   0%     0 /  0%     1 /100%     1
2 IP_segment_2              Num    0% 100%   0%     0 /  0%     1 /100%     1
3 IP_segment_3              Num    0% 100%   0%     0 /  0%     1 /100%     1
4 IP_segment_4              Num    0% 100%   0%     0 /  0%     1 /100%     1
5 class                     Nom  100%   0%   0%     0 /  0%     1 /100%     1

2015-09-06 20:28:42 AnomalyDetectionTask [INFO] Statistics:

  Nodes always inlier: 5 (12,2%)
  Nodes always outlier: 18 (43,9%)
  Nodes both inlier and outlier: 18 (43,9%)
  (Sum: 41)

  Total range queries: 41
  Max memory usage: 37 MB
  Total process time: 0,09 ms

2015-09-06 20:28:42 AnomalyDetectionTask [INFO] Information of dataset:

Relation Name:  rb_flow_dataset_1
Num Instances:  1
Num Attributes: 2

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 src_country_code          Str  100%   0%   0%     0 /  0%     1 /100%     1
2 class                     Nom  100%   0%   0%     0 /  0%     1 /100%     1

2015-09-06 20:28:42 AnomalyDetectionTask [INFO] Statistics:

  Nodes always inlier: 22 (53,7%)
  Nodes always outlier: 0 (0,0%)
  Nodes both inlier and outlier: 19 (46,3%)
  (Sum: 41)

  Total range queries: 21
  Max memory usage: 32 MB
  Total process time: 0,01 ms
```

Figura 4-3. Visualización del log donde se aprecia al detector en funcionamiento

Podemos ver que en la figura anterior tenemos monitorizado los primeros outliers e inliers, dado que estamos monitorizando IPs y país de origen podemos observar que mientras en el país de origen tenemos ya asimilados los outliers, en la dirección IPv4 aún no se termina de identificar todas las IPv4 por lo que siguen siendo objetos extraños ante el detector.

Esto es debido a que una IPv4 está compuesta por 4 segmentos, la monitorización de esos segmentos es lo que provoca la lenta convergencia del detector, dado que tenemos un rango de direcciones IPv4 a monitorizar.

Una vez que el productor ha emitido más de 100000 instancias el detector de anomalías muestra otras estadísticas:

```
2015-09-06 20:37:19 AnomalyDetectionTask [INFO] Information of dataset:

Relation Name:  rb_flow_dataset_0
Num Instances:  1
Num Attributes: 5

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 IP_segment_1              Num   0% 100%   0%     0 /  0%     1 /100%     1
2 IP_segment_2              Num   0% 100%   0%     0 /  0%     1 /100%     1
3 IP_segment_3              Num   0% 100%   0%     0 /  0%     1 /100%     1
4 IP_segment_4              Num   0% 100%   0%     0 /  0%     1 /100%     1
5 class                     Nom 100%   0%   0%     0 /  0%     1 /100%     1

2015-09-06 20:37:19 AnomalyDetectionTask [INFO] Statistics:

  Nodes always inlier: 104118 (100,0%)
  Nodes always outlier: 0 (0,0%)
  Nodes both inlier and outlier: 46 (0,0%)
  (Sum: 104164)

  Total range queries: 151
  Max memory usage: 427 MB
  Total process time: 1,45 ms

2015-09-06 20:37:19 AnomalyDetectionTask [INFO] Information of dataset:

Relation Name:  rb_flow_dataset_1
Num Instances:  1
Num Attributes: 2

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 src_country_code          Str 100%   0%   0%     0 /  0%     1 /100%     1
2 class                     Nom 100%   0%   0%     0 /  0%     1 /100%     1

2015-09-06 20:37:19 AnomalyDetectionTask [INFO] Statistics:

  Nodes always inlier: 104145 (100,0%)
  Nodes always outlier: 0 (0,0%)
  Nodes both inlier and outlier: 19 (0,0%)
  (Sum: 104164)

  Total range queries: 21
  Max memory usage: 427 MB
  Total process time: 0,64 ms
```

Figura 4-4. Visualización del log con las estadísticas del detector tras 100000 instancias

Hasta el momento el detector evoluciona favorablemente y sigue su procesamiento con las instancais que envía el productor. Supongamos que se produce un evento, que contiene una anomalía, para ser más concretos, supongamos que aparece una dirección IP que no está registrada en la ventana del detector:

```
{"dst_country_code":"US", "engine_id_name":"IANA-L4", "application_id_name":"APP B",
"biflow_direction":"initiator", "dst":"152.85.134.221", "client_rssi_num":-41,
"ip_protocol_version":4, "first_switched":1441564260, "type":"NetFlowv10", "dot11_status":"PROBING",
"pkts":373, "building":"Building A", "l4_proto":8, "src_net_name":"0.0.0.0/0",
"flow_end_reason":"idle timeout", "client_mac_vendor":"SAMSUNG ELECTRO-MECHANICS",
"src_country_code":"US", "src_net":"0.0.0.0/0", "dst_net":"0.0.0.0/0", "floor":"Floor C",
"sensor_ip":"90.1.44.3", "direction":"ingress", "timestamp":1441564260, "wireless_id":"SSID_A",
"client_mac":"00:11:22:33:44:22", "dst_net_name":"0.0.0.0/0", "src":"192.168.1.43", "campus":"Campus
E", "client_latlong":"37.89,26.39", "application_id":"5:21", "src_port":80, "sensor_uuid":4,
"engine_id":19, "bytes":2276, "flow_sampler_id":0, "wireless_station":"00:00:00:00:00:25",
"http":"http://www.apple.com", "namespace_uuid":2222, "sensor_name":"sensorA"}
```


Para emitir este evento lo lanzamos de forma manual a través de la consola productora de Kafka. Si revisamos el registro de la tarea. Podemos observar que el detector, tras computar el nuevo objeto, a obtenido un nuevo resultado en las estadísticas:

```
2015-09-06 21:52:00 AnomalyDetectionTask [INFO] Information of dataset:

Relation Name:  rb_flow_dataset_2
Num Instances:  1
Num Attributes: 5

     Name                    Type  Nom  Int Real     Missing      Unique  Dist
1 IP_segment_1              Num    0% 100%   0%     0 /  0%     1 /100%     1
2 IP_segment_2              Num    0% 100%   0%     0 /  0%     1 /100%     1
3 IP_segment_3              Num    0% 100%   0%     0 /  0%     1 /100%     1
4 IP_segment_4              Num    0% 100%   0%     0 /  0%     1 /100%     1
5 class                     Nom 100%   0%   0%     0 /  0%     1 /100%     1

2015-09-06 21:52:00 AnomalyDetectionTask [INFO] Statistics:

  Nodes always inlier: 127174 (100,0%)
  Nodes always outlier: 1 (0,0%)
  Nodes both inlier and outlier: 28 (0,0%)
  (Sum: 127174)

  Total range queries: 79
  Max memory usage: 475 MB
  Total process time: 0,26 ms

2015-09-06 21:52:00 AnomalyDetectionTask [INFO] Detected outlier in event :
{"dst_country_code":"US","engine_id_name":"IANA-L4","application_id_name":"APP B","biflow_direction":
"initiator","dst":"152.85.134.221","client_rssi_num":"-41","ip_protocol_version":"4","first_switched"
:"1441564260","type":"NetFlowv10","dot11_status":"PROBING","pkts":"373","building":"Building A","l4_p
roto":"8","src_net_name":"0.0.0.0/0","flow_end_reason":"idle timeout","client_mac_vendor":"SAMSUNG EL
ECTRO-MECHANICS","src_country_code":"US","src_net":"0.0.0.0/0","dst_net":"0.0.0.0/0","floor":"Floor C
","sensor_ip":"90.1.44.3","direction":"ingress","timestamp":"1441564260","wireless_id":"SSID_A","clie
nt_mac":"00:11:22:33:44:22","dst_net_name":"0.0.0.0/0","src":"192.168.1.43","campus":"Campus E","clie
nt_latlong":"37.89,26.39","outlier":"IP_segment_1","application_id":"5:21","src_port":"80","sensor_uu
id":"4","engine_id":"19","bytes":"2276","flow_sampler_id":"0","wireless_station":"00:00:00:00:00:25",
"http":"http://www.apple.com","namespace_uuid":"2222","sensor_name":"sensorA","outlier":"dst"}
```

Figura 4-5. Visualización del log con la detección de una anomalía

Efectivamente el detector de anomalías detecta una dirección anómala no registrada y es clasificada como outlier enriqueciendo el evento. De esta forma hemos probado que MOA procesa los eventos en tiempo real y que el detector ejerce su función de forma favorable.

5 CONCLUSIONES

Tras la realización de este proyecto se ha podido comprobar en pequeña escala a lo que hoy día se enfrenta las grandes empresas, como aquellas que trabajan en los medios sociales (Facebook, Twitter, LinkedIn) así como aquellas que velan por la seguridad de nuestra información como es redBorder.

No cabe duda de que esta pequeña introducción al universo Big data ha sido muy productiva, el entendimiento y comprensión de las herramientas ha permitido simular el entorno de redBorder para poder hacer las pruebas pertienente con MOA.

El detector de anomalías desarrollado con MOA tiene su potencial sin embargo tras varias pruebas se han visto desventajas como la falta de escalabilidad por parte de MOA. No obstante el proyecto aquí presentado puede ser interesante para pequeñas empresas o para sistemas que no manipulen grandes volúmenes de datos. Otra desventaja que se ha visto frente al volumen de datos es que MOA puede ocasionar un cuello de botella al no ser escalable y que el detector puede tener incosistencia a tener múltiples detectores en ejecución sin una centralización de los datos.

Por supuesto, se han visto reforzado los estudios de la carrera en lo referente a lenguajes de programación y se han aprendido otros nuevos conceptos que junto con los ya sedimentado han potenciado mis capacidades y aptitudes. Este proyecto también ha servido para adentrarme en el mundo de machine learning, una rama muy interesante y que me llama mucho la atención.

5.1 Puntos de mejora

Dado a la creciente ola de datos que estar por venir es evidente que MOA no es la mejor opción para afrontar dichos volúmenes de datos. Sin embargo pueden intentarse mejorar su fiabilidad y eficiencia realizando ajuste en los flujos de datos de Kafka como la fusión de estos o la asignación a partir de un atributo único como las direcciones MAC de los clientes.

No obstante cabe recordar que hay un proyecto en desarrollo que es la implementación distribuida de MOA conocido como SAMOA. Dicho proyecto, aunque es aún de escasa utilidad para la detección de anomalías, está arrojando buenos resultados sobre el comportamiento de los algoritmos distribuidos.

Aun con todo lo anterior debemos de evaluar las posibilidades de crear detectores con otras aplicaciones conocidas y diseñadas para tales propósitos como Apache Spark que posee una librería de algoritmos distribuidos sobre machine learning, además al tratarse de una proyecto apache posee una buena integración de con sus otras herramientas. H2O, utilizado por otras grandes corporaciones como Paypal, es otra herramienta interesante que emplea redes neuronales para la detección de anomalías basándose en el error a la hora de reconstruir las instancias.

6 PRESUPUESTO

En este apartado presentamos un breve presupuesto donde podemos ver los materiales utilizados así como las horas de trabajo necesarias para la adquisición de los conocimientos necesarios, puesta en marcha y evaluación del proyecto.

PRESUPUESTO			
Recursos Humanos			**23.994,10 €**
CONCEPTO	**Coste**	**Horas**	**Total**
Ingeniero de telecomunicaciones			
Septiembre 2014 a Noviembre 2014	35,00 €	30	1.050,00 €
Febrero 2015 a Junio 2015	35,00 €	225	7.875,00 €
Julio 2015 a Septiembre 2015	35,00 €	185	6.475,00 €
Recursos Materiales			**8.594,10 €**
CONCEPTO	**Coste**	**Unidades**	**Total**
Cisco UCS C260 M2 (Anfitrión virtualización)	6.794,10 €	1	6.794,10 €
Mac Book Pro (Mid 2010)	1.800,00 €	1	1.800,00 €
TOTAL			32.588,20 €

REFERENCIAS

[1] Tom White - Hadoop - The Definitive Guide [Libro]

Editorial : O'Reilly Media ISBN : 978-1-4919-0163-2

[2] Nishant Garg - Learning Apache Kafka [Libro]

Editorial : Packt Publishing ISBN : 978-1-7843-9309-0

[3] Flavio Junqueira & Benjamin Reed - ZooKeeper [Libro]

Editorial : O'Reilly Media ISBN : 978-1-449-36130-3

[4] Evan Stubbs - Big Data Bit Innovation [Libro]

Editorial : WILEY ISBN : 978-1-118-72464-4

[5] Jason Bell - Machine Learning [Libro]

Editorial : WILEY ISBN : 978-1-119-88906-0

[6] Jared Dean - Big Data, Data Mining and Machine Learning

Editorial : WILEY ISBN : 978-1-118-61804-2

[7] Albert Bifet, Geoff Holmes, Richard Kirkby & Bernhard Pfahronger - MOA Data Stream Mining [En línea] [Documento PDF]

Disponible : http://heanet.dl.sourceforge.net/project/moa-datastream/documentation/StreamMining.pdf

[8] Albert Bifet, Richard Kirkby, Philipp Kranen & Peter Reutemann - MOA Manual [En línea] [Documento PDF]

Disponible : http://heanet.dl.sourceforge.net/project/moa-datastream/documentation/Manual.pdf

[9] Apache Software Foundation - Apache Hadoop [En línea] [Web]

Disponible : http://hadoop.apache.org/docs/current/

[10] Apache Software Foundation - Apache Kafka [En línea] [Web]

Disponible : http://kafka.apache.org

[11] Apache Software Foundation - Apache ZooKeeper [En línea] [Web]

Disponible : http://zookeeper.apache.org

[12] Apache Software Foundation - Apache Samza [En línea] [Web]

Disponible : http://samza.apache.org

[13] The University of Waikato - Documentation of WEKA [En línea] [Web]

Disponible : http://www.cs.waikato.ac.nz/ml/weka/documentation.html

[14] Dimitrios Georgiadis, Maria Kontaki, Anastasios Gounaris, Apostolos Papadopoulos, Kostas Tsichlas & Yannis Manolopoulos - Continuous Outlier Detection in Data Streams [En línea] [Documento PDF]

Disponible : https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCoQFjAAahUKEwipm7uanOvHAhXEOxoKHUX0Azc&url=http%3A%2F%2Fdelab.csd.auth.gr%2Fpapers%2FSIGMOD2013gkgptm.pdf&usg=AFQjCNFDzJbLEQeKG3WAgRILL_7DYVOABQ&sig2=_B0JTseYOQFS7

MVhO-Nqcg

[15] Maria Kontaki, Anastasios Gounaris, Apostolos Papadopoulos, Kostas Tsichlas & Yannis Manolopoulos - Cotinuous Monitoring of Distance-Based Outliers over Data Streams [En línea] [Documento PDF]

Disponible :
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5767923&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5767923

ANEXO A : ZOOKEEPER – INSTALACIÓN Y EJECUCIÓN

En este anexo mostraremos la instalación y ejecución de un servicio ZooKeeper, es necesaria la instalación de este primer servicio ya que permitirá que Kafka sincronice sus subscriptores con los topics.

Lo primero que necesitamos hacer es descargar la actual versión de ZooKeeper, se puede hacer a través de la siguiente URL:

```
http://zookeeper.apache.org
```

Puede descargarse la versión más reciente de ZooKeeper en el siguiente enlace:

```
http://apache.rediris.es/zookeeper/stable/
```

Una vez descargada la versión más reciente debemos descomprimir empleando para ello el siguiente comando donde x-y-z es la versión de ZooKeeper que se ha descargado:

```
$ > tar –xvf zookeeper–x–y–z.tar
```

Una vez descomprimida podemos ver que hay varios directorios en el directorio /bin podemos encontrar los ejecutables de ZooKeeper:

```
/bin
  |- zkCleanup.sh
  |- zkCli.sh
  |- zkEnv.sh
  |- zkServer.sh
```

De todos los scripts el que inicia y detiene el servidor es zkServer.sh, pero para ello es necesario darle una configuración inicial, tenemos una configuración de ejemplo en el directorio /conf denominada zoo_sample.cfg nosotros la usaremos para nuestro caso de uso propuesto en este proyecto, pues no necesitamos configuraciones más avanzadas:

/conf/zoo_sample.cfg

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
```

Una vez que tenemos la configuración simplemente hay que inicializar el servidor, pero antes debemos de renombrar el fichero de configuración a zoo.cfg, ya que por defecto buscará ese fichero ante la ejecución del siguiente comando:

```
$ > bin/zkServer.sh start
```

Si todo ha ido de forma correcta verá un mensaje por terminal que indicará que el servidor se está ejecutando de forma correcta.

ANEXO B : KAFKA – INSTALACIÓN Y EJECUCIÓN

En este anexo mostraremos la instalación y ejecución de un servicio Kafka, para ejecutar Kafka es necesario haber instalado previamente el servicio de ZooKeeper y tenerlo en ejecucicón para que Kafka pueda sincronizar los subscriptores y registrar los topics.

Al igual que con otros servicios de Apache, lo primero que tenemos que hacer es descargar la versión más reciente, podemos encontrarla en la siguiente URL

```
http://kafka.apache.org
```

Para la comodidad del lector a continuación pongo el enlace de la versión que había a la hora de escribir esta memoria:

```
https://www.apache.org/dyn/closer.cgi?path=/kafka/0.8.2.1/kafka_2.10-0.8.2.1.tgz
```

A continuación descargamos el comprimido y lo descomprimimos con el siguiente comando:

```
$ > tar –xvf Kafka_x.y_0.8.2.1
```

Donde x.y corresponde a la versión de Scala con la que se ha compilado, una vez en el directorio podemos encontrar otros dos principales que son /bin y /config. En el directorio /bin encontramos los scripts para la ejecución del servicio así como algunas utilidades para crear los topics y administrar. Dentro del directorio /config encontramos el fichero de configuración del servicio (server.properties), si echamos un vistazo al contenido podemos ver la configuración por defecto:

/config/server.properties

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
# see kafka.server.KafkaConfig for additional details and defaults

############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

############################# Socket Server Settings #############################

# The port the socket server listens on
port=9092

# Hostname the broker will bind to. If not set, the server will bind to all interfaces
#host.name=localhost

# Hostname the broker will advertise to producers and consumers. If not set, it uses the
# value for "host.name" if configured.  Otherwise, it will use the value returned from
# java.net.InetAddress.getCanonicalHostName().
#advertised.host.name=<hostname routable by clients>

# The port to publish to ZooKeeper for clients to use. If this is not set,
# it will publish the same port that the broker binds to.
#advertised.port=<port accessible by clients>

# The number of threads handling network requests
num.network.threads=3

# The number of threads doing disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

# A comma seperated list of directories under which to store log files
log.dirs=/tmp/kafka-logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at startup and flushing at shutdown.
# This value is recommended to be increased for installations with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Log Flush Policy #############################
```

```
# Messages are immediately written to the filesystem but by default we only fsync() to sync
# the OS cache lazily. The following configurations control the flush of data to disk.
# There are a few important trade-offs here:
#    1. Durability: Unflushed data may be lost if you are not using replication.
#    2. Latency: Very large flush intervals may lead to latency spikes when the flush does occur as there will be a lot of data to flush.
#    3. Throughput: The flush is generally the most expensive operation, and a small flush interval may lead to exceessive seeks.
# The settings below allow one to configure the flush policy to flush data after a period of time or
# every N messages (or both). This can be done globally and overridden on a per-topic basis.

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before we force a flush
#log.flush.interval.ms=1000

############################# Log Retention Policy #############################

# The following configurations control the disposal of log segments. The policy can
# be set to delete segments after a period of time, or after a given size has accumulated.
# A segment will be deleted whenever *either* of these criteria are met. Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned from the log as long as the remaining
# segments don't drop below log.retention.bytes.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is reached a new log segment will be created.
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they can be deleted according
# to the retention policies
log.retention.check.interval.ms=300000

# By default the log cleaner is disabled and the log retention policy will default to just delete segments after their retention expires.
# If log.cleaner.enable=true is set the cleaner will be enabled and individual logs can then be marked for log compaction.
log.cleaner.enable=false

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000
```

En nuestro caso no es necesario revisar la configuración de los productores o consumidores, dado que hemos implementado un programa en Java para ello. Una vez que tenemos la configuración definida procederemos a levantar el servidor de Kafka (recuerde levantar el servidor de ZooKeeper antes de llevar a cabo esta acción), para ello hacemos uso del siguiente comando:

```
$ > bin/kafka-server-start.sh config/server.properties
```

Si observamos el árbol que se ha definido en el servidor de ZooKeeper podemos observar algunos znodes de interés como es brokers o consumers. Para poder ver estas carpetas debe de hacer uso del Script zkCli.sh, consulte la ayuda disponible para más detalle sobre su uso.

A continuación crearemos un topic de prueba para comprobar la conectividad para ello ejecutaremos el siguiente script:

```
$ > bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic rb_flow
--replication-factor 1 --partitions 1
```

Si todo ha ido bien se le mostrará el mensaje “Created topic <topic>” Como puede observarse hemos establecido los parámetros de factor de replicación y de particionado a 1, esto ha sido así debido a que no es objetivo de este proyecto poner en marcha un sistema orientado a Big data y para hacer de una forma más simple este paso. El topic se llamará “rb_flow”.

Procederemos a hacer una prueba de funcionalidad, para ello ejecutaremos en un terminal el siguiente script con sus parámetros correspondientes:

```
$ > bin/kafka-console-producer.sh --topic rb_flow --broker-list localhost:9092
```

Si observamos los parámetros que recibe es el topic en el que va a escribir y el listado de brokers que están asociados a dicho topic, de esa forma la información es enviada de forma correcta.

En otro terminal aparte ejecutamos el siguiente script:

```
$ > bin/kafka-console-consumer.sh --topic rb_flow --zookeeper localhost:2181
```

Observando los parámetros podemos ver que uno de ellos coincide con el del productor que es el topic desde el cual leer, el otro parámetro está relacionado con zookeeper, le indicamos la máquina y el puerto al que comunicarse para la sincronización, cuando un consumidor quiere subscribirse utiliza zookeeper para encontrar el topic al que conectar y sincronizar los datos con éste.

A continuación sólo deberemos de escribir un texto en el terminar del productor y este texto se verá reflejado en el del consumidor, de esta manera sabremos que todo ha ido de forma correcta.

ANEXO C : YARN – INSTALACIÓN Y EJECUCIÓN

Siguiendo la línea de los anteriores anexos pasaremos a ver la instalación y ejecución de YARN, recordemos que YARN forma parte de la base sobre la que se apoya Samza y que por lo tanto es indispensable su ejecución para poder lanzar las tareas de Samza.

Lo primero que debemos de hacer es descargar la versión más reciente del servicio de Hadoop, para ello podemos descargarla de la siguiente URL:

```
http://hadoop.apache.org
```

Nuevamente para la comodidad del lector se ofrece el enlace a la versión más reciente:

```
http://apache.rediris.es/hadoop/common/hadoop-2.7.1/
```

Una vez descargado el archivo lo descomprimimos con el siguiente comando

```
$ > tar –xvf hadoop-2.7.1.tar
```

Una vez descomprimido cambiamos al directorio de hadoop y ejecutamos los siguientes comandos:

```
$ > sbin/yarn-daemon.sh start resourcemanager
$ > sbin/yarn-daemon.sh start nodemanager
```

Una vez que tenemos el entorno funcionando simplemente debemos de desarrollar nuestra tarea de Samza y subirla con los correspondientes comandos.

ANEXO D : TAREA SAMZA – DATAMINING (JAVA)

AnomalyDetectionTask.java

```
package net.redborder.samza.tasks;

import Exceptions.ConfigurationException;
import RBClasses.RBInstance;
import moa.cluster.Cluster;
import moa.cluster.Clustering;
import moa.cluster.SphereCluster;
import moa.clusterers.outliers.MCOD.MCOD;
import moa.clusterers.outliers.MyBaseOutlierDetector;
import moa.gui.visualization.DataPoint;
import org.apache.samza.config.Config;
import org.apache.samza.system.IncomingMessageEnvelope;
import org.apache.samza.system.OutgoingMessageEnvelope;
import org.apache.samza.system.SystemStream;
import org.apache.samza.task.*;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.type.TypeReference;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import weka.core.Attribute;
import weka.core.Instance;
import weka.core.Instances;

import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class AnomalyDetectionTask implements StreamTask, InitableTask{

    private static final SystemStream OUTPUT_STREAM = new SystemStream("kafka",
"outliers_detecteds");

    /**
     * Con este objeto podremos hacer loggin y seguir las acciones del programa
     */
    private static final Logger log =
LoggerFactory.getLogger(AnomalyDetectionTask.class);

    /**
     * Con este objeto podremos asociar los flujos que recibe el detector con el
conjunto de datos que utilizara
     * cada vez que se llegue al limite pasaremos a comprobarlo.
     */
    private Map<String, LinkedList<Instances>> instanciasFlujo = new HashMap<String,
LinkedList<Instances>>();

    /**
     * Con este objeto podrmeos asociar a cada flujo de entrada su salida
correspondiente
     */
    private Map<String, SystemStream> salidaSistema = new HashMap<String,
SystemStream>();

    /**
```

```
     * Con este objeto podremos mapear los detectores a los que pertenece cada flujo,
de manera que podremos enviar a cada
     * uno las instancias correspondientes a cada flujo con el objeto
"instanciasFlujo"
     */
    private Map<String, Map<String, MCOD>> detectorFlujo = new HashMap<String,
Map<String,MCOD>>();

    /**
     * Con este objeto podremos manipular la instancia actual del mensaje del flujo
recibido, identificando previamente
     * este flujo.
     */
    private Instances datasetActualMOA = null;

    /**
     * Objeto que permite obtener el mapeo de los JSON
     */
    private ObjectMapper _mapper = new ObjectMapper();

    /**
     * Cadena que nos permitira identificar el flujo del que proviene el mensaje
recibido
     * se va actualizando por cada mensaje que se recibe.
     */
    private String flujoRecibido = "<Flujo sin especificar>";

    /**
     * Con esta variable tendremos una marca temporal de la instancia recibida
     */
    private int timeStamp = 0;

    /**
     * Con esta variable tendremos el número de muestras total procesadas
     */
    private int procesadas = 0;

    /**
     * Con este objeto tendremos disponible un buffer para el almacenamiento de los
DataPoints
     */
    private LinkedList<DataPoint> buffer = new LinkedList<DataPoint>();

    /**
     * Con esta variable especificamos el límite de instancias procesadas para
mostrar la información
     */
    private int totalProcesadas = 3000;

    /**
     * Con esta variable podemos comprobar si MOA ha sido configurado o no
     */
    private boolean configurado = false;

    /**
     * Con esta variable podemos determinar el número de muestras utilizadas para
entrenar
     */
    private long totalEntrenamiento = 10000;

    /**
     * Con esta variable determinaremos cuando el detector entrena y cuando no
     */
    private boolean entreno = true;

    @Override
    public void process(IncomingMessageEnvelope envelope, MessageCollector collector,
```

```
TaskCoordinator coordinator) throws Exception {

        // Obtenemos el nombre del flujo
        flujoRecibido = envelope.getSystemStreamPartition().getStream();

        // Comprobamos si el mensaje que hemos recibido es de configuracion, de ser
asi
        if(instanciasFlujo.containsKey(flujoRecibido)) {  // En caso contrario
aplicamos el algoritmo al evento que se obtenga si se ha creado previamente


            // Con este objeto podremos obtener el JSON parseado y manipularlo de
forma mas comoda
            // Creamos un mapeo de clave/valor para el evento que obtenemos del
mensaje
            // Basicamente lo que estamos haciendo es coger nuestro JSON y hacerle un
mapeado para que de esa manera
            // tengamos una acceso mas simple a los valores.
            Map<String, Object> evento = (Map<String, Object>)envelope.getMessage();

            // Obtenemos el conjunto de datasets asociados al flujo
            LinkedList<Instances> datasetsObtenidos =
instanciasFlujo.get(flujoRecibido);

            // Obtenemos los detectores asociados al flujo
            Map<String, MCOD> detectoresAsociados = detectorFlujo.get(flujoRecibido);

            // Para cada dataset
            for(int i = 0; i < datasetsObtenidos.size(); i++){
                // Obtenemos del conjunto de datasets el dataset que actualmente
usaremos
                datasetActualMOA = datasetsObtenidos.get(i);

                // Creamos la instancia
                RBInstance instancia = new
RBInstance(datasetActualMOA.numAttributes());

                // Le asignamos el dataset
                instancia.setDataset(datasetActualMOA);

                // Escribimos el JSON que hemos obtenido como evento
                instancia.setEvent(_mapper.writeValueAsString(evento));

                // Comprobamos el número de atributos de clase que tiene la
estrucutra de la instancia
                // Si tenemos un único atributo entonces obtenemos ese atributo

                if(instancia.numClasses() == 1){

                    // Obtenemos la única clase existente en la instancia
                    String clase = instancia.classAttribute().value(0);

                    if(evento.containsKey(clase)){
                        // Obtenemos el atributo
                        String atributo = evento.get(clase).toString();

                        // Si el atributo es de tipo IP
                        if(atributo.matches("\\d{1,3}(\\.\\d{1,3}){3}")){

                            // Convertimos el literal en octetos
                            int[] octetos = ip2int(atributo);

                            // Asignamos los valores de instancia
                            for(int j = 0; j < instancia.numAttributes(); j++){

                                if(j < 4)
                                    instancia.setValue(j, octetos[j]);
                                else
```

```
                                    instancia.setValue(j, clase);
                                }
                                // Si el atributo es numérico
                            }else if(atributo.matches("\\d+")){
                                if(instancia.numValues() == 2) {
                                    instancia.setValue(0, Double.valueOf(atributo));
                                    instancia.setValue(1, clase);
                                }
                                // Si el atributo es una cadena
                            }else if(atributo.matches(".+")){
                                if(instancia.numValues() == 2){
                                    instancia.setValue(0, atributo);
                                    instancia.setValue(1, clase);
                                }
                            }

                            // agregamos al dataset la nueva instancia
                            datasetActualMOA.add(instancia);
                            // Obtenemos el detector basandonos en el nombre de la
relacion del dataset
                            MCOD detector =
detectoresAsociados.get(datasetActualMOA.relationName());
                            // Entrenamos al detector con la instancia
                            detector.trainOnInstanceImpl(instancia);

                            if(datasetActualMOA.size() > totalEntrenamiento ){
                                entreno = false;
                            }

                            // Una vez alcanzado el limite del dataset
                            if(!entreno && datasetActualMOA.size() % totalProcesadas ==
0) {

                                log.info("Information of dataset:\n\n" +
datasetActualMOA.toSummaryString());

                                // Obtenemos las estadisticas
                                log.info(detector.getStatistics());

                                // Comprobamos si hemos detectado anomalías
                                if(!detector.getOutliersResult().isEmpty()){

                                    // Para cada outlier
                                    for(MyBaseOutlierDetector.Outlier o :
detector.getOutliersResult()){

                                        // Obtenemos su instancia de referencia
                                        RBInstance instanciaAnomala = (RBInstance)o.inst;


                                        // Obtenemos el JSON del evento de la instancia
                                        Map<String, String> json =
_mapper.readValue(instanciaAnomala.getEvent(), new TypeReference<HashMap<String,
String>>(){});


                                        // Enriquecemos con el nuevo campo

json.put("outlier",instanciaAnomala.classAttribute().name());

                                        log.info("Detected outlier in event : \n" +
_mapper.writeValueAsString(json));

                                        // Lo enviamos al sistema
                                        collector.send(new
OutgoingMessageEnvelope(OUTPUT_STREAM, json));
```

```
                                }
                            }


                            // Limpiamos el dataset
                            datasetActualMOA.clear();

                        }

                    }

                }

            }

        }

    /**
     * Método que convierte un literal de IP en octetos
     * @param ipAddress Dirección de IP literal
     * @return Array con los octetos de IP
     */
    public int[] ip2int(String ipAddress){
        // Declaramos los octetos enteros
        int[] octetos = {0, 0, 0, 0};
        // Obtenemos los octetos literales dividiendo la cadena
        String[] octets = ipAddress.split("\\.");

        int i = 3;
        // Por cada obteto
        for(String octet : octets){
            // Obtenemos su formato numérico
            octetos[i] = Integer.parseInt(octet);
            i--;
        }

        // Devolvemos los obtetos obtenidos en el array
        return octetos;

    }

    @SuppressWarnings("unchecked")
    @Override
    public void init(Config conf, TaskContext context) throws Exception {
        // Permitimos los campos que no estan delimitado por comillas dobles
        _mapper.configure(JsonParser.Feature.ALLOW_UNQUOTED_FIELD_NAMES, true);
        // Permitimos los campos que estan delimitados por comillas simples
        _mapper.configure(JsonParser.Feature.ALLOW_SINGLE_QUOTES, true);

        // Si no está configurado
        if(!configurado){

            // Obtenemos el número de pasadas antes de la revisión
            if(conf.containsKey("outlier.review")) {
                totalProcesadas = Integer.parseInt(conf.get("outlier.review"));
                log.info("Set review at : " + totalProcesadas);
            }

            if(conf.containsKey("outlier.train")){
                totalEntrenamiento = Integer.parseInt(conf.get("outlier.train"));
                log.info("Set train at : " + totalEntrenamiento);
            }

            // Obtenemos el listado de flujos de entrada
            List<String> inputStreams = conf.getList("outlier.input");
```

```
            // Para cada flujo del listado
            // Comprobamos si tiene definida las clases
            for(String stream : inputStreams) {

                log.info("Stream configuration : " + stream);

                // Si existen clases definidas para el flujo
                if (conf.containsKey("outlier." + stream + ".classes")) {

                    // Contador para conocer el número de datasets
                    int numDataset = 0;

                    // Obtenemos el listado de clases para el flujo definido
                    List<String> classes = conf.getList("outlier." + stream +
".classes");
                    // Lista enlazada de datasets
                    LinkedList<Instances> datasets = new LinkedList<Instances>();
                    // Mapeado de detectores
                    Map<String, MCOD> arrayDetectores = new HashMap<String, MCOD>();

                    // Tomaremos las propias clases como atributos de cadena, de
forma que
                    // cada clase es en sí un atributo de cadena
                    for (String cls : classes) {
                        log.info("Added class : " + cls);

                        // Listado de atributos
                        ArrayList<Attribute> atributos = new ArrayList<Attribute>();

                        // Si existen atributos para la clase definida
                        if (conf.containsKey("outlier." + stream + ".class." + cls))
{

                            List<String> attDefined = conf.getList("outlier." +
stream + ".class." + cls);

                            for (String att : attDefined) {

                                // Con este patrón nos aseguramos de obtener los
atributos y sus respectivos valores
                                Matcher m =
Pattern.compile("(?<name>[_\\w][\\w\\d]*):(?<type>string|numeric|IP)").matcher(att);

                                // Si coincide
                                if (m.matches()) {
                                    // Obtenemos el nombre del atributo
                                    String name_att = m.group("name");
                                    // Obtenemos el tipo del atributo
                                    String type_att = m.group("type");

                                    // Si es de tipo string
                                    if (type_att.equals("string")) {
                                        atributos.add(new Attribute(name_att,
(ArrayList) null));

                                        log.info("Added attribute:<Type> : " + att);
                                        // Si es de tipo numérico
                                    } else if (type_att.equals("numeric")) {
                                        atributos.add(new Attribute(name_att));
                                        log.info("Added attribute:<Type> : " + att);
                                        // Si es de tipo "IP"
                                    } else if (type_att.equals("IP")) {

                                        // Para cada segmento
                                        for (int i = 1; i <= 4; i++)
                                            atributos.add(new Attribute("IP_segment_"
+ i));
```

```
                                    log.info("Added attribute:<Type> : " + att);
                                    // Si no es de ningún tipo anterior
                                }else{
                                    log.error( m.group("type") + " is not a valid
type! Attribute \"" + name_att + "\" not added!");
                                }

                            }

                        }// fin bucle for de atributos

                    } else {
                        // Añadimos un nuevo atributo con el nombre del campo del
flujo
                        atributos.add(new Attribute(cls, (ArrayList) null));
                        log.info("Added attribute:<Type> : " + cls + ":string");
                    }

                    // Clases en las que se clasificará el atributo
                    ArrayList<String> clasesNominales = new ArrayList<String>();
                    // Añadimos la clase a al grupo de clases nominales
                    clasesNominales.add(cls);

                    // Añadimos un nuevo atributo de clases con el nombre del
campo del flujo
                    atributos.add(new Attribute("class", clasesNominales));

                    log.info("Added class attribute with content : " +
clasesNominales.toString());

                    // Creamos un nuevo dataset
                    Instances dataset = new Instances(stream + "_dataset_" +
numDataset++, atributos, 1000);
                    // Establecemos el índice del atributo de clases
                    dataset.setClassIndex(atributos.size() - 1);

                    log.info("Created dataset : \n\n" +
dataset.toSummaryString());

                    // Agregamos el nuevo dataset
                    datasets.add(dataset);

                    // Creamos un nuevo detector
                    MCOD detector = new MCOD();

                    log.info("Created new outlier detector");

                    // Si se han definido los parámetros del detector
                    if (conf.containsKey("outlier." + stream + "." + cls +
".parameters")) {

                        log.info("Setting outlier detector parameters");
                        // Obtenemos los parámetros del detector
                        List<String> parametros = conf.getList("outlier." +
stream + "." + cls + ".parameters");


                        for (String option : parametros) {

                            String[] opt = option.split(":");

                            if (opt[0].matches("window")) {

detector.windowSizeOption.setValue(Integer.valueOf(opt[1]));
                            } else if (opt[0].matches("radius")) {

detector.radiusOption.setValue(Double.valueOf(opt[1]));
                            } else if (opt[0].matches("k")) {
```

```
detector.kOption.setValue(Integer.valueOf(opt[1]));
                                } else {
                                    log.error(opt[0] + " is not valid parameter!");
                                }
                            }
                        }else{
                            throw new ConfigurationException("Don't definied
outlier." + stream + "."+ cls + ".parameters");
                        }

                        // Lo preparamos para el uso
                        detector.prepareForUse();

                        log.info("Outlier detector prepared with parameters: \n "
                                + "\twindow: " +
detector.windowSizeOption.getValue() + "\n"
                                + "\tradius: " + detector.radiusOption.getValue() +
"\n"
                                + "\tk: " + detector.kOption.getValue() + "");

                        log.info("Adding outlier detector with dataset : " +
dataset.relationName());
                        // Agregamos el detector al dataset
                        arrayDetectores.put(dataset.relationName(), detector);

                    }

                    log.info("Adding datasets with stream : " + stream);
                    instanciasFlujo.put(stream, datasets);
                    log.info("Adding array of outlier detector with stream : " +
stream);
                    detectorFlujo.put(stream, arrayDetectores);

                    log.info("Finish configuration of MOA for stream : \"" + stream +
"\"");
                } else {
                    // En caso contrario lanzamos una excepción de configuración
                    throw new ConfigurationException("Don't defined outlier." +
stream + ".classes");
                }
            }

            configurado = true;

        }

    }

    /**
     * Metodo que permite monitorizar las instancias de forma individual
     * @param instancia Instancia a inspeccionar de forma individual
     */
    public void inspeccionarConDetectorIndividualmente(Instance instancia) {

        // Obtenemos el array de detectores
        Map<String, MCOD> arrayDetectores = detectorFlujo.get(flujoRecibido);

        // COmprobamos si tenemos uno o mas detectores
        if (arrayDetectores.size() >= 1) {

            // Obtenemos el detector actual para dicha clase
            MCOD detectorActual = arrayDetectores.get(instancia.dataset());
            // Entrenamos el detector
            detectorActual.trainOnInstanceImpl(instancia);
            log.info(detectorActual.getStatistics());
```

```
            // Asignamos un espacio temporal
            timeStamp++;
            // Obtenemos el numero de muestras procesadas
            procesadas++;

            // Creamos un nuevo punto para esa instancia en un tiempo fijo de tiempo
            DataPoint point = new DataPoint(instancia, timeStamp);
            // Almacenamos el punto en el buffer
            buffer.add(point);

            // Si superamos el limite de ventana
            if (buffer.size() > 1000) {
                // entonces eliminamos el primer elemento (que sera el mas antiguo)
                buffer.removeFirst();
            }

            // Creamos un nuevo cluster
            Clustering gtCluster = null;

            // Para cada punto de dato del buffer
            for (DataPoint dp : buffer) {
                // Actualizamos si peso basandonos en el espacio de tiempo asignado
                dp.updateWeight(timeStamp, 1000);
                // Y creamos un nuevo cluster con dicha informacion
                gtCluster = new Clustering(new ArrayList<DataPoint>(buffer));
            }

            log.info("Cluster size : " + gtCluster.size() + " cluster/s");

            // Para cada cluster que contiene
            for (Cluster c : gtCluster.getClustering()) {

                SphereCluster sc = (SphereCluster) c;

                log.info("[CLASS]RADIUS GT : " + sc.getRadius());
                log.info("[CLASS]WEIGH GT : " + sc.getWeight());

            }

        } else {
            log.info("Don't exist references for \"" + flujoRecibido + "\"");
        }
    }

    /**
     * este metodo permite normalizar el conjunto de datos, de esta forma podemos
aplicar escalado con
     * valores comprendidos entre 0 y 1. Esto es necesario si la dimension de los
valores son muy desproporcionadas.
     * @param dataset Conjunto de instancias que se va a normalizar
     */
    public void normalizarDataset(Instances dataset){

        // Creamos un minimo con el valor mas alto de un Double
        double min = Double.POSITIVE_INFINITY;
        // Creamos el maximo con el valor mas pequeño de un Double
        double max = Double.NEGATIVE_INFINITY;
        // Diferencia entre el maximo y el minimo sera 0.0
        double diffMaxMin = 0.0;

        for (int i = 0 ; i < dataset.numInstances(); i++){
            // Para cada instancia del dataset
            Instance instancia = dataset.get(i);
            for (int j = 0; j < instancia.numAttributes(); j++) {

                // Obtenemos el valor de sus atributos
                double value = instancia.value(j);
                // Obtenemos el minimo valor de todo el dataset
```

```
                if(value < min)
                    min = value;
                // Obtenemos el maximo valor de todo el dataset
                if(value > max)
                    max = value;
            }
        }

        // Una vez obtenido el maximo y el minimo efectuamos la diferencia de ambos
        diffMaxMin = max - min;

        for (int i = 0; i < dataset.numInstances(); i++){
            // Para cada instancia del dataset
            Instance instancia = dataset.get(i);

            for (int j = 0; j < instancia.numAttributes() ; j++) {
                // Comprobamos que la diferencia sea distinto de 1 y de 0
                if(diffMaxMin != 1 && diffMaxMin != 0){
                    // Obtenemos el valor de la instancia que hemos obtenido
                    double v = instancia.value(j);
                    // Hacemos la diferencia con el minimo valor que se ha obtenido y
lo dividimos entre
                    // la diferencia del maximo y el minimo
                    v = (v - min)/diffMaxMin;
                    // Establecemos el nuevo valor para la instancia que hemos
procesado
                    instancia.setValue(j, v);
                }
            }// For de atributos de instancia

        }// For de instancias

    }


}
```

ANEXO E : CONFIGURACIÓN - SAMZA Y PRODUCTOR SINTÉTICO

datamining.properties

```
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with redBorder. If not, see <http://www.gnu.org/licenses/>.

# Job
job.factory.class=org.apache.samza.job.yarn.YarnJobFactory
job.name=datamining

# YARN
yarn.package.path=file:///opt/rb/var/rb-samza-moa/app/rb-samza-moa.tar.gz
yarn.container.count=1
yarn.queue=samza

# Task
task.class=net.redborder.samza.tasks.AnomalyDetectionTask
task.opts=-Dsamza.application.name=datamining
task.drop.deserialization.errors=true


task.inputs=kafka.rb_flow
task.checkpoint.factory=org.apache.samza.checkpoint.kafka.KafkaCheckpointManagerFactory
task.checkpoint.system=kafka
task.checkpoint.replication.factor=1

# Serializers
serializers.registry.string.class=org.apache.samza.serializers.StringSerdeFactory
serializers.registry.json.class=org.apache.samza.serializers.JsonSerdeFactory

# Kafka System
systems.kafka.samza.factory=org.apache.samza.system.kafka.KafkaSystemFactory
systems.kafka.samza.key.serde=string
systems.kafka.samza.msg.serde=json
systems.kafka.consumer.zookeeper.connect=localhost:2181
systems.kafka.producer.bootstrap.servers=localhost:9092

# Outlier Detector
outlier.input=rb_flow
outlier.review=1
outlier.rb_flow.classes=src, src_country_code, dst
outlier.rb_flow.class.src=src:IP
outlier.rb_flow.class.dst=dst:IP
outlier.rb_flow.class.src_country_code=src_country_code:string
outlier.rb_flow.src.parameters=window:10000, radius:10, k:8
outlier.rb_flow.dst.parameters=window:10000, radius:10, k:10
outlier.rb_flow.src_country_code.parameters=window:100, radius:0.1, k:20
```

configProducer.yml

```
zk_connect: localhost:2181
topic: rb_flow
threads: 2
rate: 100000
fields:
  client_latlong:
    type: constant
    value: '37.89,26.39'
  dst_country_code:
    type: constant
    value: US
  dot11_status:
    type: collection
    values:
      - ASSOCIATED
      - PROBING
      - UNKNOWN
  bytes:
    type: integer
    min: 10
    max: 3000
  src_net_name:
    type: constant
    value: 0.0.0.0/0
  flow_sampler_id:
    type: constant
    value: 0
  direction:
    type: constant
    value: ingress
  wireless_station:
    type: mac
    min: 00:00:00:00:00:11
    max: 00:00:00:00:00:44
  biflow_direction:
    type: constant
    value: initiator
  pkts:
    type: integer
    min: 1
    max: 500
  dst:
    type: ip
    min: 80.82.34.12
    max: 80.82.34.72
  type:
    type: constant
    value: NetFlowv10
  campus:
    type: collection
    values:
      - Campus A
      - Campus B
      - Campus C
      - Campus D
      - Campus E
  building:
    type: collection
    values:
      - Building A
      - Building B
      - Building C
      - Building D
      - Building E
  floor:
```

```
    type: collection
    values:
      - Floor A
      - Floor B
      - Floor C
      - Floor D
      - Floor E
  timestamp:
    type: timestamp
  client_mac:
    type: mac
    min: 00:11:22:33:44:00
    max: 00:11:22:33:44:FF
  wireless_id:
    type: collection
    values:
      - SSID_A
      - SSID_B
      - SSID_C
  flow_end_reason:
    type: constant
    value: idle timeout
  src_net:
    type: constant
    value: 0.0.0.0/0
  client_rssi_num:
    type: integer
    negative: true
    min: 0
    max: 80
  engine_id_name:
    type: constant
    value: IANA-L4
  src:
    type: ip
    min: 192.168.1.1
    max: 192.168.1.128
  application_id:
    type: composition
    separator: ':'
    components:
      - type: integer
        min: 0
        max: 11
      - type: integer
        min: 0
        max: 101
  sensor_ip:
    type: constant
    value: 90.1.44.3
  application_id_name:
    type: collection
    values:
      - APP A
      - APP B
      - APP C
      - APP D
      - APP E
  dst_net:
    type: constant
    value: 0.0.0.0/0
  dst_net_name:
    type: constant
    value: 0.0.0.0/0
  l4_proto:
    type: integer
    min: 0
    max: 11
```

```
  ip_protocol_version:
    type: constant
    value: 4
  sensor_name:
    type: collection
    values:
      - sensorA
      - sensorB
      - sensorC
  src_country_code:
    type: constant
    value: US
  engine_id:
    type: integer
    min: 0
    max: 21
  client_mac_vendor:
    type: constant
    value: SAMSUNG ELECTRO-MECHANICS
  first_switched:
    type: timestamp
  http:
    type: collection
    values:
      - 'http://www.google.es'
      - 'http://www.microsoft.es'
      - 'http://www.apple.com'
  src_port:
    type: constant
    value: 80
  namespace_uuid:
    type: collection
    values:
      - 1111
      - 2222
      - 3333
  sensor_uuid:
    type: collection
    values:
      - 2
      - 7
      - 4
```