U-tad
CENTRO UNIVERSITARIO DE TECNOLOGÍA Y ARTE DIGITAL

Técnicas de Machine Learning en Spark Streaming

Proyecto final de curso 05/06

Experto en Big Data

Autor: Antonio Soriano Tolosa

Introducción

Procesamiento en streaming y Machine Learning

Muchas fuentes de datos representan series de eventos que se producen de forma continua en el tiempo (por citar casos, logs, transacciones, sensores, etc.). **Tradicionalmente**, estos eventos se agrupaban de forma periódica en ***batchs*** *o lotes*, salvándose para luego ser procesadas en conjunto. Esta forma de proceder introducía latencias innecesarias entre la generación de los datos y la obtención de resultados, así como asumía de forma implícita que los datos deben estar "completos" en algún momento y, por lo tanto, pueden ser usados en ese instante para realizar predicciones precisas. A partir de esta forma de procesar por lotes, el siguiente paso lógico en analíticas es aprovechar la naturaleza continua de la generación de datos para procesarlos de la misma manera: como flujos de datos o streams. El **streaming** viene a ser una nueva forma de pensar en la infraestructura de datos.

El ***machine learning*** o aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos. En muchas ocasiones el campo de actuación del aprendizaje automático se solapa con el de la estadística. Sin embargo, el aprendizaje automático se centra más en el estudio de la complejidad computacional de los problemas, muchos de ellos de clase *NP-hard* (difíciles de resolver en un orden de tiempo polinómico), por lo que gran parte de la investigación realizada en aprendizaje automático está enfocada al diseño de soluciones factibles a esos problemas.

Tradicionalmente el ***machine learning*** ha estado muy relacionado con el tipo de procesado *batch*. Con la aparición de nuevos frameworks de computación distribuida, se ha disparado una corriente investigativa para adaptar y utilizar algoritmos de *machine learning* en sistemas de procesamiento en streaming.

El objetivo de este proyecto es estudiar la aplicabilidad de técnicas de *machine learning* ante streams de datos, centrándose este caso en métodos de segmentación o clusterización de datos utilizando la tecnología Spark Streaming.

Técnicas de clustering

El análisis de clústeres o **clustering** es una técnica de **aprendizaje no supervisado**, ya que se busca encontrar relaciones entre una serie de variables descriptivas y no la que guardan éstas con respecto a una variable objetivo.

Esta técnica tiene como objetivo **segmentar** en diferentes grupos (llamados **clústeres**) una serie de **objetos** definidos por un conjunto de características. Con la segmentación se busca que los objetos que pertenezcan a un mismo clúster posean **características similares y diferenciadoras** con respecto a los que pertenecen al resto de clústeres.

$$\mathcal{O} = \{o_1, o_2, \dots, o_N\} \;\rightarrow\; \mathcal{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N),\;\; \boldsymbol{x}_n \in \mathbb{R}^d$$

Así, se dispone de un conjunto de objetos $\mathcal{O}$ definidos por una serie de características recogidas en un vector *d-dimensional* $\boldsymbol{x} \in \mathbb{R}^d$, y se desea agruparlos en K grupos o clústeres $\mathcal{C} = \{C_1, \dots, C_K\}$ de forma que los objetos de cada clúster compartan propiedades comunes.

Las técnicas de clustering tienen aplicación en multitud de áreas, como por ejemplo en biología, medicina, marketing, teoría de señal, visión por computador, seguros, análisis web, biometría… Las técnicas de clustering también son usadas como paso previo a otras técnicas de minería de datos y aprendizaje automático, como por ejemplo realizar un análisis exploratorio inicial, como etapa de pre-procesado para eliminar outliers, reducir datos, inicialización de variables…

K-Means: Algoritmo de Lloyd

Uno de los **métodos de clustering** más utilizado es **K-Means** [1], un método de agrupamiento que tiene como objetivo la partición de un conjunto de *N* observaciones en *K* grupos, en el que cada observación se asigna al grupo cuya media de datos (centroide del cluster) este más próxima.

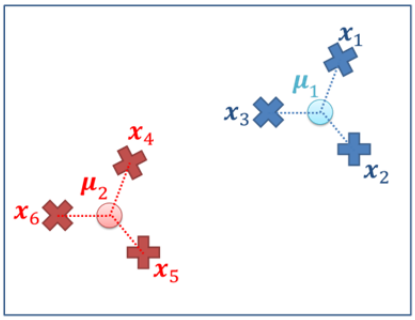
Observaciones: objetos definidos por un conjunto de características, representadas éstas por un vector real *d-dimensional*, $\mathcal{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N)$, $\boldsymbol{x}_n \in \mathbb{R}^d$.

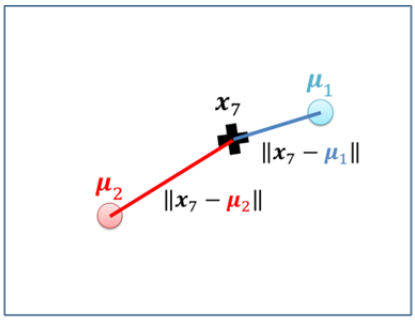
Cada una de las observaciones se asigna a uno de los posibles **clusters**, $\mathcal{C} = \{C_1, \dots, C_K\}$:

$$C_i = \left\{\boldsymbol{x}_j : \|\boldsymbol{x}_j - \boldsymbol{\mu}_i\|^2 \leq \|\boldsymbol{x}_j - \boldsymbol{\mu}_k\|^2 \; \forall 1 \leq k \leq K\right\} \qquad C_i :\Leftrightarrow \boldsymbol{\mu}_i = \frac{1}{|C_i|}\sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j \qquad |C_i| = n_i$$

donde $\boldsymbol{\mu}_i$ hace referencia al **centroide** de los puntos $\boldsymbol{x}_j \in C_i$ y n_i al número de puntos asignados a C_i.

Figura 1 – Modelo de clustering propuesto por el método K-Means.





$$C_1 \equiv \left\{Data: \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \;\; Centroid: \;\; \mu_1 = \frac{1}{3}(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3)\right\}$$
$$C_2 \equiv \left\{Data: \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6 \;\; Centroid: \;\; \mu_2 = \frac{1}{3}(\boldsymbol{x}_4 + \boldsymbol{x}_5 + \boldsymbol{x}_6)\right\}$$

$$\|\boldsymbol{x}_7 - \mu_1\|^2 \leq \|\boldsymbol{x}_7 - \mu_2\|^2 \;\Leftrightarrow\; \boldsymbol{x}_7 \in C_1$$

Figura 2 – Ejemplo de un modelo de clustering K-Means.

El objetivo es definir los clusters de manera que se minimice la **variación total intra-cluster** o **distorsión** (*within-clusters sum of square, **WCSS***), definida como el total de las sumas de las distancias al cuadrado entre todos los puntos pertenecientes a cada uno de los clusters:

WCSS, variación total intra-clusters, medida relativa a la compactación conseguida con el clustering:

$$D_k = \sum_{\boldsymbol{x}_i \in C_k} \sum_{\boldsymbol{x}_j \in C_k} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = 2n_k \sum_{\boldsymbol{x}_i \in C_k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2 \qquad WCSS = \sum_{k=1}^{K} \frac{1}{2n_k} D_k$$

$\boldsymbol{D_k}$, suma de las distancias *intra-cluster* al cuadrado entre los $|C_k| = n_k$ puntos pertenecientes al cluster C_k.

Función objetivo, minimizar la variación total intra-cluster (también llamada distorsión):

$$\underset{\mathcal{C}}{\operatorname{argmin}}\, J(\mathcal{X}, \mathcal{C}) = \underset{\mathcal{C}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in C_k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2$$

Figura 3 – Problema de optimización planteado en K-Means para el entrenamiento del modelo.

El **problema de optimización** definido por el método K-Means es computacionalmente difícil (**NP-hard**). Para un número de clusters K, dimensión de los datos d y n observaciones el problema puede ser resuelto de forma exacta en un tiempo $O\left(n^{dk+1}logn\right)$ [2].

Para la resolución de este problema se suelen emplear eficientes heurísticas que requieren menor coste computacional y convergen más rápidamente. El algoritmo *K-Means* más común, conocido como **algoritmo de Lloyd** [3], utiliza una técnica de refinamiento iterativo basada en descenso por gradiente para la resolución del problema de optimización y así calcular los centroides del modelo:

1. **Inicializar** los centroides asociados a cada cluster: $\boldsymbol{\mu}_1{}^{(t=0)}$, $\boldsymbol{\mu}_2{}^{(t=0)}$, … , $\boldsymbol{\mu}_K{}^{(t=0)}$
2. Refinamiento **iterativo**: repetir hasta que se consiga convergencia
   - **Asignación** de cada observación a un cluster:
   $$C_i{}^{(t)} = \left\{ \boldsymbol{x}_j : \left\|\boldsymbol{x}_j - \boldsymbol{\mu}_i{}^{(t)}\right\|^2 \leq \left\|\boldsymbol{x}_j - \boldsymbol{\mu}_k{}^{(t)}\right\|^2 \quad \forall 1 \leq k \leq K \right\}$$
   - **Actualización** de los centroides:
   $$\boldsymbol{\mu}_i{}^{(t+1)} = \frac{1}{\left|C_i{}^{(t)}\right|} \sum_{\boldsymbol{x}_j \in C_i{}^{(t)}} \boldsymbol{x}_j$$
   - Chequear convergencia.

Figura 4: Pseudocódigo del algoritmo de Lloyd.

Existen diversas técnicas para **inicializar los centroides del modelo**. Dos de los métodos más utilizados [4] son la selección aleatoria de *K* observaciones entre los datos de entrada como centroides (conocido como **método de Forgy**) o el **particionado aleatorio**, asignando de forma aleatoria un cluster a cada una de las observaciones y calculando los centroides obtenidos con este particionado.

La función de coste que define K-Means no es convexa, por lo que la utilización del algoritmo de Lloyd no garantiza que la convergencia se realice hacia un mínimo global; el algoritmo de Lloyd es susceptible de alcanzar un **óptimo local**. Suele ser habitual **lanzar el algoritmo varias veces** usando **diferentes valores** en la **inicialización** de los **centroides**, escogiendo como **modelo final** aquel que proporcione un valor de **distorsión menor**.

El tiempo de ejecución del algoritmo de Lloyd se suele definir como $O(nkdi)$. Tanto la calidad de la solución obtenida, como el número de iteraciones i necesarias para que el algoritmo converja (y por lo tanto el tiempo de ejecución) dependen de la elección de los centroides iniciales y de la distribución de los datos a segmentar. Para distribuciones de datos que presentan una marcada estructura en clusters, el número de iteraciones i para lograr convergencia suele ser bajo, considerando el algoritmo de complejidad "lineal" $O(n)$.

Existen implementaciones del algoritmo que utilizan diversas técnicas para mejorar su eficiencia; por ejemplo *K-Means++* [5] utiliza un método de inicialización más complejo que ayuda a que el algoritmo converja a una mejor solución. Otras implementaciones intentan mejorar el *speed-up* de cada iteración del algoritmo teniendo en cuenta que después de varias iteraciones los centroides no suelen moverse de forma muy abrupta; muchos de puntos asignados a un cluster en una iteración, seguirán perteneciendo al mismo cluster o a algún otro cercano en la siguiente iteración, por lo que no siempre es necesario volver a recalcular las distancias de estos puntos con todos los centroides.

K-Means requiere especificar a priori el número de clusters a usar. La selección automática del número de clusters es algo ambigua, dependiendo de la distribución de los datos y del criterio del usuario (homogeneidad en el número de muestras por cluster, clusters con extensión espacial similar…). Podemos encontrar diversas heurísticas para la selección automática de este parámetro [6]–[8]

Mini-Batch K-Means

Como ya se ha comentado anteriormente, el método de clustering *K-Means* se basa en la minimización de una función de coste denominada distorsión o variación total intra-cluster. La solución exacta de este problema de optimización es computacionalmente muy costosa, por lo que se suelen utilizar otras técnicas basadas en diferentes heurísticas que mejoran la eficiencia y convergen de forma más rápidamente a un mínimo local. El algoritmo de Lloyd, basado en un descenso por gradiente, es ampliamente utilizado para realizar un clustering basado en *K-Means*.

El **algoritmo de Lloyd** se define como un **algoritmo *batch***, ya que, en cada uno de los pasos del refinamiento iterativo de los centroides que propone, se hace necesario de disponer de todo el conjunto de observaciones para operar con ellas.

En ciertas aplicaciones en las que se necesita clusterizar un conjunto muy elevado de observaciones el algoritmo de Lloyd puede resultar muy lento (incluso utilizando implementaciones que incorporan diferentes optimizaciones con respecto al algoritmo base) o puede ser inviable por el requerimiento de disponer de todas las observaciones para realizar el procesado en cada iteración. En estos casos se puede utilizar una variante del algoritmo denominada ***mini-batch K-means*** [9] (basada en un descenso por el gradiente utilizando mini-batches).

La variante mini-batch *K-Means* trata de optimizar la misma función objetivo propuesta por el método *K-Means*, pero a diferencia del algoritmo de Lloyd, en cada iteración solo utiliza un subconjunto aleatorio de las observaciones, reduciendo la cantidad de operaciones computacionales y de almacenamiento necesarias para converger a una solución local; en contrapartida, los resultados obtenidos son, por lo general, un poco peores que los obtenidos con el algoritmo batch.

Al igual que en el algoritmo batch, los centroides asociados a cada cluster necesitan ser inicializados. Una vez inicializados, los centroides son refinados de forma iterativa, utilizando en cada iteración un subconjunto aleatorio de observaciones (formando un mini-batch, $N_{mini-batch} \ll N_{total}$) y realizando en cada iteración dos pasos diferenciados:

1º - Con los datos del mini-batch se realiza un proceso similar al usado en el algoritmo batch:

- Cada observación del mini-batch se asigna a un determinado cluster basado en la distancia a los centroides $\boldsymbol{c}_i,\ 1 \leq i \leq K$. A cada cluster se le asignan m_i observaciones del mini-batch.
- Se calculan los **centroides asociados a los datos del mini-batch** $\boldsymbol{x}_i,\ 1 \leq i \leq K$como la media de los puntos del mini-batch pertenecientes a cada cluster.

2º - Los centroides $\boldsymbol{c}_i$ son actualizados utilizando una media móvil respecto al número de observaciones con los centroides del mini-batch $\boldsymbol{x}_i$; n_i observaciones habían sido asignadas hasta el momento al cluster definido por $\boldsymbol{c}_i$; m_i observaciones han sido asignadas a los centroides $\boldsymbol{x}_i$ en esta iteración:

$$\boldsymbol{c}_{t+1} = \frac{\boldsymbol{c}_t * n_t + \boldsymbol{x}_t * m_t}{n_t + m_t}$$

Se actualizan el número de observaciones asignadas hasta ahora a cada cluster: $n_{t+1} = n_t + m_t$

A continuación incluimos un simple ejemplo para ilustrar el funcionamiento del algoritmo:

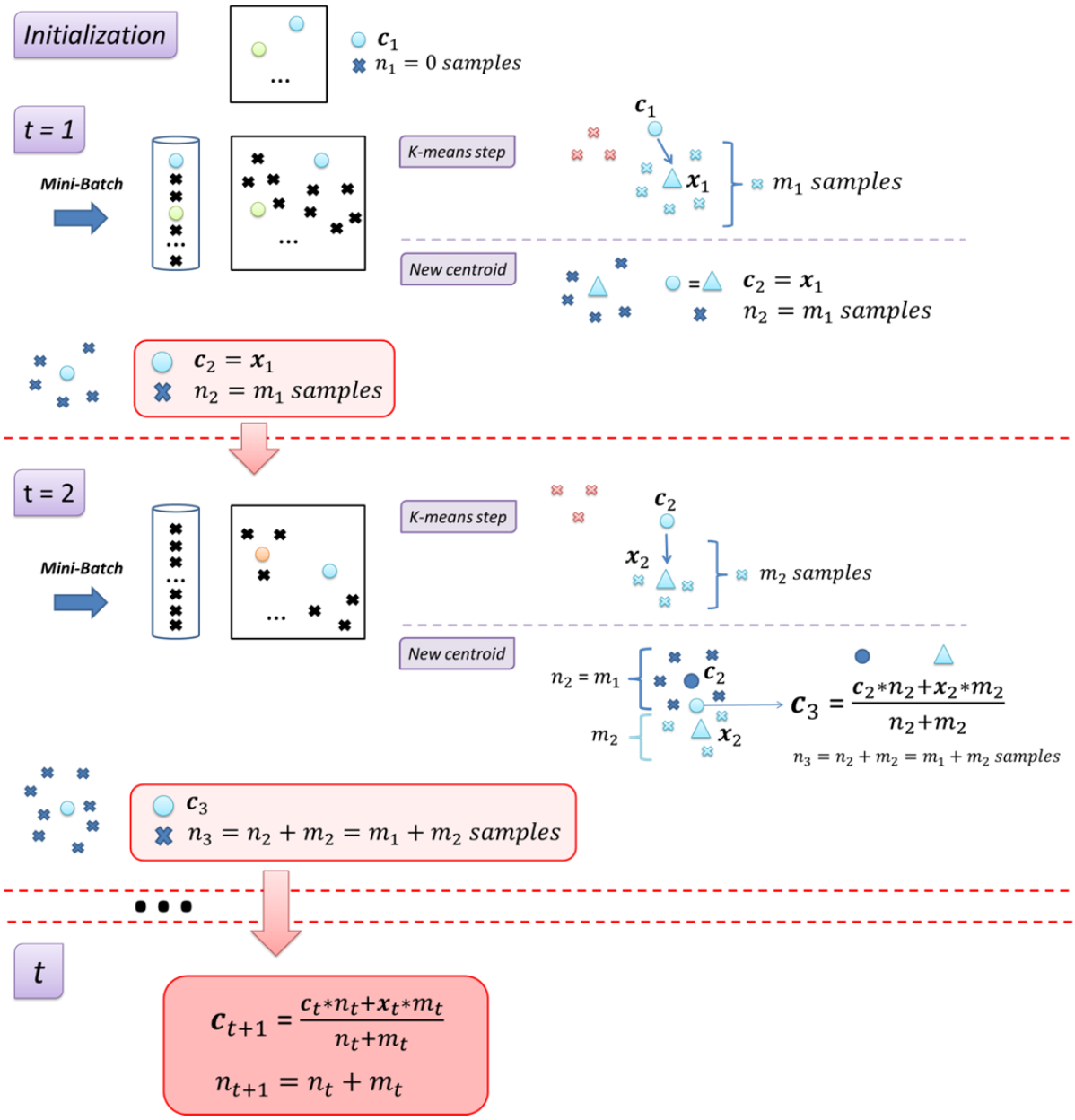


Figura 5 – Ejemplo ilustrativo del funcionamiento del algoritmo mini-batch K-Means

Clustering en Apache Spark

Apache Spark [10] es un motor para el procesado distribuido de datos a gran escala en un sistema compuesto por un cluster de máquinas. Incorpora una serie de librerías que proporcionan herramientas de alto nivel como son Spark SQL para el procesado de datos estructurados utilizando el lenguaje SQL, GraphX para procesado basado en grafos o Spark Streaming para el procesado online.

En cuanto a algoritmos y técnicas relacionadas con ***Machine Learning*** incorpora una librería denominada **MLlib**. En esta librería encontrarnos dos implementaciones de algoritmos de clustering basados en el método K-Means: K-Means|| y streaming K-Means.

MLlib: K-Means||

MLlib incorpora un algoritmo de **clustering batch** basado en una **versión paralela y optimizada del algoritmo de Lloyd**, incorporando una **técnica de inicialización** denominada **K-Means||.**

Como ya comentamos, la inicialización del algoritmo de Lloyd es crucial en cuanto al rendimiento del algoritmo, tanto en tiempo de procesado, como en la calidad de la solución obtenida. En una variante del algoritmo denominada **K-Means++** [5] se utiliza una **técnica especial de inicialización** de los clústeres para que se encuentren diseminados de tal forma que, al aplicar el algoritmo iterativo, el resultado al que converja tienda a ser mejor al logrado si se inicializasen de forma aleatoria. Aunque la técnica de selección de centroides iniciales requiere de tiempo extra, la parte de refinamiento iterativo se mejora con la elección de esos centroides, tendiendo a converger más rápidamente y a una solución mejor.

La técnica de inicialización utilizada en K-Means++ requiere procesar todos los datos en *K* pasadas secuenciales para calcular una buena aproximación de los centroides, lo cual puede limitar su utilización cuando el conjunto de observaciones es muy elevado. En [11] se propone una mejora de esta técnica de inicialización denominada K-Means|| en la que, utilizando procesado en paralelo, se reduce el número de pasadas necesarias.

El algoritmo de clustering batch K-Means que incorpora MLlib nos permite seleccionar el método de inicialización K-Means|| o una inicialización aleatoria a la hora de escoger los centroides iniciales.

El algoritmo de refinamiento iterativo está programado de forma que el procesado en cada una de las iteraciones esta optimizado para realizarse en paralelo de forma distribuida. Además, incorpora la posibilidad de lanzar simultáneamente el entrenamiento de varios modelos con diferentes inicializaciones (definiendo el número en el parámetro *runs*), devolviéndonos el modelo que alcance el mejor resultado (aquél que obtenga un menor valor de distorsión).

MLlib: Streaming K-Means

La librería MLlib incorpora también una versión del algoritmo K-Means para ser utilizada cuando los datos son capturados en streaming. Esta versión del algoritmo ha sido diseñada para ser utilizada utilizando **Spark Streaming**, una extensión del core de Spark, que permite el procesado escalable y con tolerancia a fallos de streams con alto flujo de datos en tiempo real. Así, los clusters pueden ser estimados de forma dinámica, actualizándose los centroides conforme van llegando nuevos datos.

Este algoritmo utiliza una generalización del algoritmo de mini-batch K-Means. Spark Streaming define los **DStream** para trabajar con streams de datos, representados como una **serie continua de RDDs**, cada uno de ellos conteniendo los datos que han llegado en un determinado intervalo temporal. En lo que respecta el algoritmo de clustering, cada uno de los RDDs que componen el DStream actúa como un mini-batch a la hora del entrenamiento dinámico del modelo.

Tanto en las versiones batch como mini-batch que hemos comentado del algoritmo, se asume que todas las observaciones con las que se trabajan han sido generadas por un proceso estacionario, es decir, se asume que todos los datos han sido generados con una distribución equivalente. En el caso de procesado en streaming, asumimos que los **datos pueden no ser estacionarios**, es decir, la división en clústeres de los datos puede no mantenerse constante, pudiendo evolucionar ésta con el tiempo.

Para lidiar con la no estacionaridad temporal de los datos, el algoritmo incorporado en MLlib utiliza una **variante de mini-batch K-Means**, la cual utiliza una **media móvil exponencial** en la etapa de **actualización de los centroides**. Así, se introduce un parámetro $\alpha \in [0,1]$ denominado *decay* (parámetro de olvido) para ponderar el peso que tienen los centroides calculados hasta el momento (basados en datos temporalmente más antiguos) a la hora de realizar la media móvil con los centroides calculados con los datos del último mini-batch (datos más recientes). También incluye lógica para detectar **clusters extintos**, eliminando el centroide asociado al cluster y dividiendo el cluster de mayor tamaño en dos.

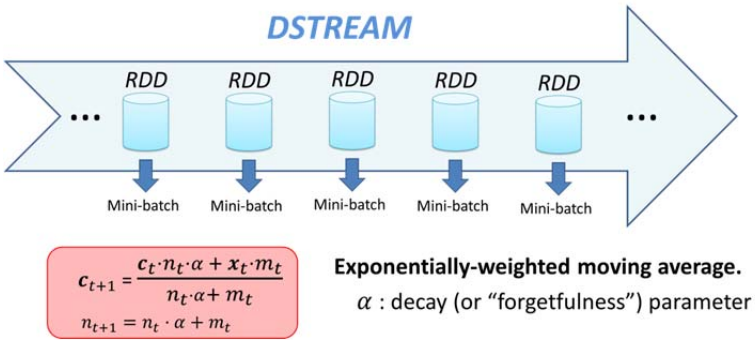


Figura 6 – Ejemplo ilustrativo del funcionamiento del algoritmo streaming K-Means.

El caso particular $\alpha = 1$ equivale algorítmicamente al procesado realizado en el algoritmo mini-batch original, en el que se supone que todos los datos son igualmente importantes a la hora de actualizar un centroide, independientemente de su antigüedad; se considera que la distribución de datos es estacionaria. Cuando $\alpha = 0$ los centroides se asumen que son los obtenidos con los datos del último mini-batch; sólo los últimos datos temporales analizados son relevantes. En la siguiente figura podemos ver el mismo ejemplo que se utilizó para ilustrar el algoritmo de mini-batch (ver figura 5), ahora en el contexto de procesado en streaming.

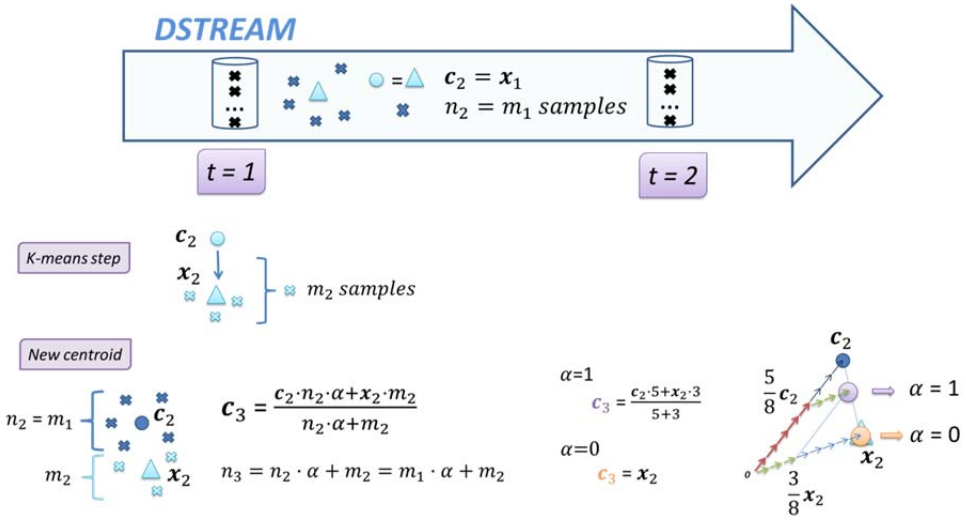


Figura 7 – Ejemplo ilustrativo del funcionamiento del algoritmo streaming K-Means.

Online K-Means - Cambio en la distribución de los datos.

El algoritmo de streaming K-Means es una variante del algoritmo mini-batch K-Means con capacidad para lidiar con un cambio en la distribución de los datos, introduciendo un parámetro α que modula la capacidad de 'olvidar' datos antiguos. Ajustando este parámetro se configura la ponderación asignada a los centroides obtenidos con datos antiguos con respecto a los calculados en el último mini-batch. Se ha creado un pequeño ejemplo práctico para ilustrar y entender mejor el funcionamiento algorítmico:

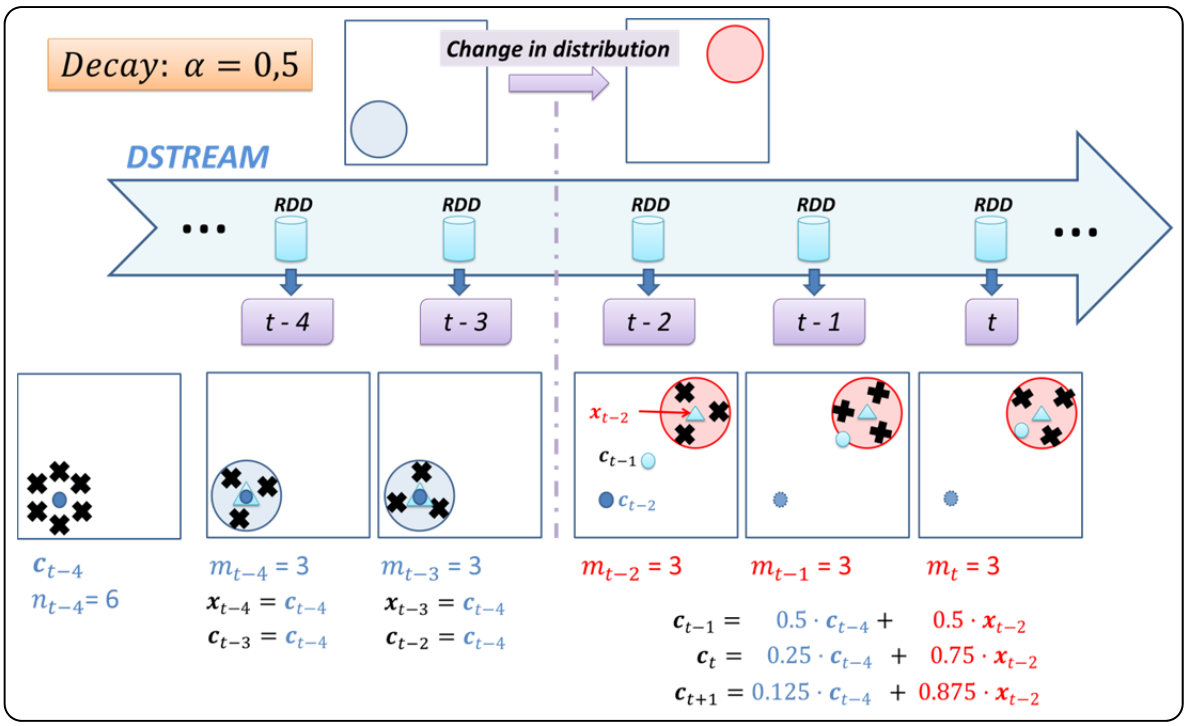


Figura 8 – Ejemplo práctico del funcionamiento del algoritmo streaming K-Means.

$$c_{t-3} = \frac{c_{t-4} \cdot n_{t-4} \cdot \alpha + x_{t-4} \cdot m_{t-4}}{n_{t-4} \cdot \alpha + m_{t-4}} =_{\{x_{t-4}=c_{t-4}\}} c_{t-4}$$

$$n_{t-3} = n_{t-4} \cdot \alpha + m_{t-4} = 6 \cdot 0.5 + 3 = 6$$

$$c_{t-2} = \frac{c_{t-3} \cdot n_{t-3} \cdot \alpha + x_{t-3} \cdot m_{t-3}}{n_{t-3} \cdot \alpha + m_{t-3}} =_{\{x_{t-3}=c_{t-3}=c_{t-4}\}} c_{t-4}$$

$$n_{t-2} = n_{t-3} \cdot \alpha + m_{t-3} = n_{t-4} \cdot \alpha^2 + m_{t-4} \cdot \alpha + m_{t-3} = 6$$

$$c_{t-1} = \frac{c_{t-2} \cdot n_{t-2} \cdot \alpha + x_{t-2} \cdot m_{t-2}}{n_{t-2} \cdot \alpha + m_{t-2}} = 0.5 \cdot c_{t-4} + 0.5 \cdot x_{t-2}$$

$$n_{t-1} = n_{t-2} \cdot \alpha + m_{t-2} = n_{t-4} \cdot \alpha^3 + m_{t-4} \cdot \alpha^2 + m_{t-3} \cdot \alpha + m_{t-2} = 3 + 3 = 6$$

$$c_t = \frac{c_{t-1} \cdot n_{t-1} \cdot \alpha + x_{t-1} \cdot m_{t-1}}{n_{t-1} \cdot \alpha + m_{t-1}} = 0.5 \cdot c_{t-1} + 0.5 \cdot x_{t-1} =_{\{x_{t-1}=x_{t-2}\}} 0.25 \cdot c_{t-4} + 0.75 \cdot x_{t-2}$$

$$n_t = n_{t-1} \cdot \alpha + m_{t-1} = 3 \cdot 0.5 + 3 \cdot 0.5 + 3 = 1.5 + 4.5 = 6$$

$$c_{t+1} = \frac{c_t \cdot n_t \cdot \alpha + x_t \cdot m_t}{n_t \cdot \alpha + m_t} = 0.5 \cdot c_t + 0.5 \cdot x_t =_{\{x_t=x_{t-2}\}} 0.125 \cdot c_{t-4} + 0.875 \cdot x_{t-2}$$

$$n_{t+1} = n_t \cdot \alpha + m_t = 1.5 \cdot 0.5 + 4.5 \cdot 0.5 + 3 = 0.75 + 5.25 = 6$$

Se ha considerado el caso en que los centroides han convergido y el algoritmo se encuentra en un estado estable. Podemos apreciar como ante un cambio en la distribución de uno de los clusters, el centroide asociado a él tiende a 'seguirlo' gracias a que el algoritmo tiende a ponderar los centroides obtenidos con datos antiguos cada vez con menor peso, asignando cada vez más peso a los nuevos datos.

Desarrollo práctico

Se han desarrollado dos módulos a modo de trabajo práctico relacionado con el presente proyecto final de curso:

- **Módulo 1:** se plantea una comparativa entre el algoritmo batch de K-Means y la variante basada en mini-batches.
- **Módulo 2**: se ha implementado un prototipo de sistema de procesado de datos en streaming, que permite la utilización del algoritmo de clustering en streaming K-Means e ilustra su funcionamiento.

Módulo 1 - Comparativa entre Batch K-Means y Mini-Batch K-Means

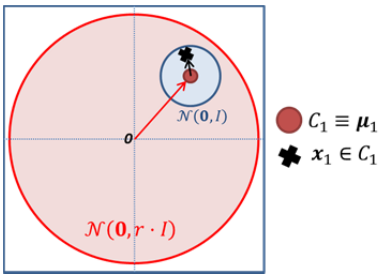
El algoritmo de procesado en streaming incorporado en Apache Spark se basa en una variante de *mini-batch K-Means*. Antes de empezar a utilizar el algoritmo con datos en streaming, se ha considerado oportuno el desarrollo de un trabajo previo con el objetivo de entender el funcionamiento del algoritmo batch base y de su variante utilizando mini-batches, y así, comenzar a utilizar y descubrir el código fuente de las implementaciones incorporadas en la librería de machine learning de Spark, poder compararlos, y entender mejor el contexto de utilización y funcionamiento de la versión en streaming.

En este módulo se ha creado una **aplicación** en el framework de procesado distribuido **Apache Spark** donde se utilizan los algoritmos incorporados en la librería MLlib para la resolución de un problema de clustering de datos basado en el modelo K-Means.

En este caso se ha pretendido resolver un mismo problema de clustering utilizando versiones batch y mini-batch del algoritmo. La aplicación tiene como objetivo principal establecer un entorno de test que nos permita entender el funcionamiento de los algoritmos, y comparar su rendimiento y eficiencia ante diferentes parámetros en el problema de clustering y configuración del entorno de procesado.

Generador de datos sintéticos

La librería MLlib incorpora en el paquete *org.apache.spark.mllib.util* la clase ***KMeansDataGenerator*** a modo de utilidad para la generación de datos sintéticos con los que testear los algoritmos de clustering. En este caso, se ha utilizado parte del código incluido en esta clase para incorporarlo en la aplicación generada.



En el modelo de generación implementado, se escogen los *K* centroides asociados a los clusters *sampleando* una distribución normal multivariante (d-dimensional) con escala r. Los datos pertenecientes a cada cluster se obtienen sumando una perturbación normal estándar d-dimensional a cada centroide.

$$C_i \equiv \boldsymbol{\mu}_i \sim \mathcal{N}(\mathbf{0}, r \cdot I) \quad \boldsymbol{x}_j \in C_i \Leftrightarrow \boldsymbol{x}_j = \boldsymbol{\mu}_i + \mathbf{z}, \;\; \mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$$

Algoritmos

Para el realizar el procesado batch se ha utilizado la implementación **K-Means||** incluida con la librería. En cuanto a la variante de procesado a partir de mini-batches, como ya comentamos, el funcionamiento algorítmico es equivalente al algoritmo de streaming K-Means para el caso particular en que el parámetro de olvido o decay $\alpha = 1$. Después del estudio del código de la implementación de ***streaming K-Means***, se ha conseguido realizar un pequeño ***wrapper*** para utilizarlo directamente en Spark con el conjunto de RDDs que se quiera, sin necesidad de utilizar un DStream asociado a Spark Straming.

Descripcion del programa

El programa implementado permite ejecutar el entrenamiento de ambos algoritmos utilizando un dataset sintético. El generador permite seleccionar el **número de observaciones** a generar, la **dimensión** de estas y el **número** de núcleos gaussianos utilizados a modo de **clusters**. Sabiendo el vector de medias asociado a cada una de las distribuciones, conocemos los centroides ideales a los cuales debería converger la solución del algoritmo de clustering.

A la hora de generar los datos, se generan troceados en diferentes RDDs (obteniendo un array de RDDs), pudiendo configurar el **número de observaciones por RDD**. Cada uno de estos RDDs actuará como **mini-batch** a la hora del entrenamiento (para el caso batch, se genera un único RDD con la unión de todos ellos).

Como ya comentamos, la calidad de la solución obtenida por estos algoritmos depende en gran medida del valor de inicialización de los centroides, por lo que suele ser habitual lanzar varios intentos y escoger aquél que obtenga un mejor resultado (aquél cuyo valor de distorsión final sea inferior). El algoritmo batch ya incorpora la posibilidad de entrenar varios modelos de forma simultánea, configurando el parámetro ***runs***. Hemos incorporado esta forma de proceder también en el código utilizado en el algoritmo de mini-batch; así, otro de los parámetros globales que podemos definir en el código es el **número de intentos** que se ejecutan en cada tanda de entrenamiento de los modelos. Las propiedades de convergencia y por lo tanto el tiempo de ejecución de los algoritmos también dependen de las condiciones iniciales, por lo que es conveniente estimar un tiempo medio de ejecución lanzando varias **tandas de entrenamiento** por cada experimento llevado a cabo.

En este caso el programa ha sido corrido en una única máquina, por lo que otro de los parámetros que se puede escoger es el **número de cores** utilizado al correrse en **modo pseudo-distribuido**. Variando el número de ejecutores utilizados podremos hacernos una idea de la explotación del paralelismo de la que hacen uso los algoritmos y las características de escalabilidad de los mismos.

Experimentos y resultados

Los experimentos han sido corridos en una única máquina utilizando el modo de ejecución pseudo-distribuido de Spark, donde se corre cada ejecutor/worker en un core. La máquina posee un procesador Intel Core i7-4770 CPU@ 3.40GHz – 8 Cores y dispone de 16 GB de memoria Ram.

En una primera prueba se han realizado un conjunto de experimentos utilizando:

- 8 cores para el procesado.
- Datos distribuidos en 10 clusters.
- 10 intentos por algoritmo.
- 10.000 muestras por mini-batch.
- Tres conjuntos de datos diferentes generados con 5, 10 y 20 dimensiones.
- Variando entre 500.000, 1.000.000, 5.000.000 y 10.000.000 el número de observaciones.
- Repitiendo cada experimento 5 veces

En total son 12 experimentos, con 5 tandas de entrenamiento de ambos algoritmos en cada experimento.

Se ha escogido el tiempo de ejecución de cada algoritmo como el valor de la mediana entre los 5 tiempos obtenidos por cada experimento. Se ha definido una medición del error cometido por cada algoritmo como la suma de la distancias de cada centroide obtenido $\boldsymbol{c}_i$ con respecto al ideal $\boldsymbol{\mu}_i$. $\varepsilon = \sum_{i=1}^{K} \|\boldsymbol{c}_i - \boldsymbol{\mu}_i\|$

En las gráficas incluidas en las figuras 8 y 9 se recogen los tiempos de ejecución y el error cometido por cada algoritmo en cada uno de los experimentos. Se puede apreciar como en el caso del algoritmo batch, la tendencia en cuanto al tiempo de ejecución (siempre hablando de un tiempo 'medio', ya sabemos que existe una marcada desviación en el tiempo de convergencia según las condiciones iniciales) puede considerarse lineal con respecto al número de muestras $O(n)$.

En el caso del algoritmo basado en mini-batches, podemos observar que el tiempo de convergencia es considerablemente menor al del caso batch cuando el número de muestras es significativamente elevado, tendiendo hacia un valor constante con respecto al número de muestras; este comportamiento es coherente, pensando que ante distribuciones de datos estacionarias y bien distribuidas entre los mini-batches, la convergencia depende del número de mini-batches utilizados (iteraciones), no del número total de datos (no se necesitan procesar todos los datos para conseguir convergencia).

Podemos ver que el algoritmo batch obtiene muy buenos resultados en cuanto a la solución obtenida, mientras que los resultados del algoritmo mini-batch suelen ser un poquito peores (hay que tener en cuenta que estamos comparándolo con respecto a un centroide ideal, si se piensa un poco se puede concluir que tampoco va a afectar en demasía al particionado final de los datos). Ante datasets masivos, aunque se utilice una versión distribuida del algoritmo batch, el tiempo de convergencia puede ser demasiado elevado en determinadas aplicaciones; en estos casos el uso del algoritmo en mini-batch puede ser más conveniente.

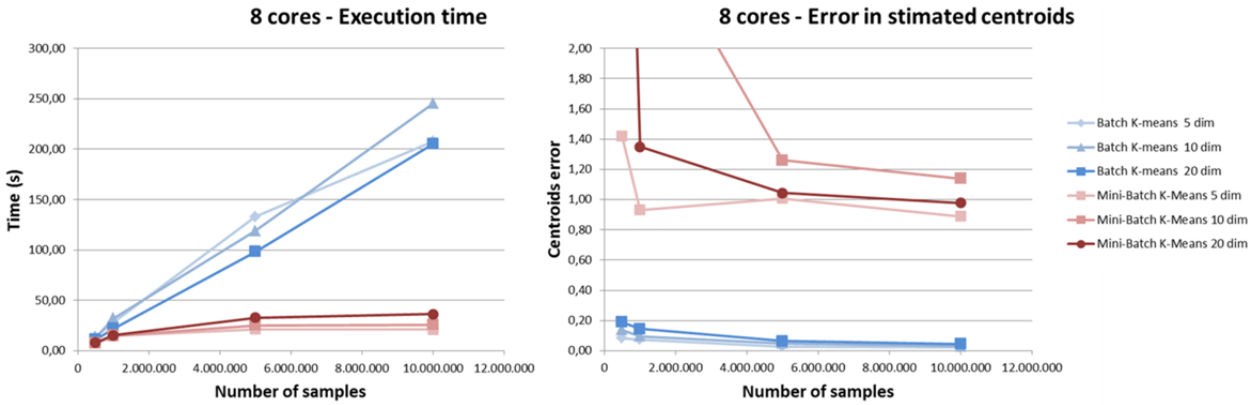


Figura 8 – Gráficas del tiempo medio de ejecución y error cometido con respecto al número de muestras.

En una segunda prueba

- Variando entre 2, 4 y 8 cores para el procesado.
- Datos distribuidos en 10 clusters.
- 10 intentos por algoritmo.
- 10.000 muestras por mini-batch.
- Un dataset de 10.000.000 observaciones con 5 dimensiones por observación.
- Repitiendo cada experimento 5 veces

En total son 4 experimentos, con 5 tandas de entrenamiento por cada uno de ellos en cada uno de los algoritmos.

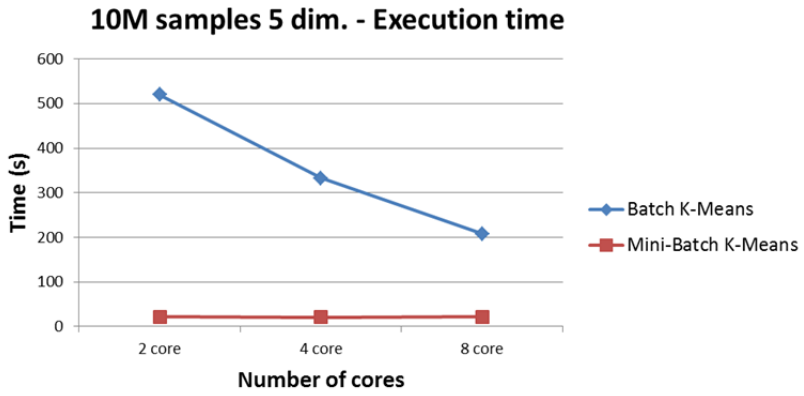


Figura 9 – Gráfica del tiempo medio de ejecución con respecto al número de cores utilizado.

Podemos observar en la gráfica como, en este caso, el algoritmo batch posee notables propiedades en cuanto a paralelismo y escalabilidad, reduciendose el tiempo de ejecucion medio con respecto al número de procesos ejecutados en paralelo utilizados en para el procesado. En el caso de utilizar 2 cores para el procesado hemos obtenido un tiempo en torno a los 500 segundos. Al repetir el experimento con 4 cores, el tiempo obtenido a estado en torno a los 300 segundos; al multiplicar por 2 el grado de paralelismo, lo deseable sería que el tiempo se redujese hacia la mitad, unos 250 segundos. En lo que respecta al algoritmo mini-batch, al estar procesando una cantidad relativamente pequeña de muestras en cada mini-batch, el efecto del paralelismo no puede apreciarse bién. Teniendo en cuenta que en cada iteración del algoritmo mini-batch se ejecuta un paso algorítmicamente idéntico al del caso batch, ante datasets masivos con tamaños de mini-batches mucho mayores, el algoritmo experimentara igualmente una tendencia en mejora de velocidad con respecto al nivel de paralelismo.

Módulo 2 - Streaming K-Means

En este módulo se ha implementado un prototipo de sistema de procesado en tiempo real que, utilizando el algoritmo '*streaming K-Means*' incorporado en MLlib, implementa un pipeline que permite realizar un clustering de streams de datos cuya distribución temporal puede ser no estacionaria.

El sistema implementado consta de dos partes diferenciadas, una de ellas encargada de la generación sintética de datos, y otra de análisis, control del procesado y visualización de resultados.

Sistema y arquitectura propuesta

En la siguiente figura se muestra un esquema con la arquitectura y tecnologías usadas en el sistema propuesto:

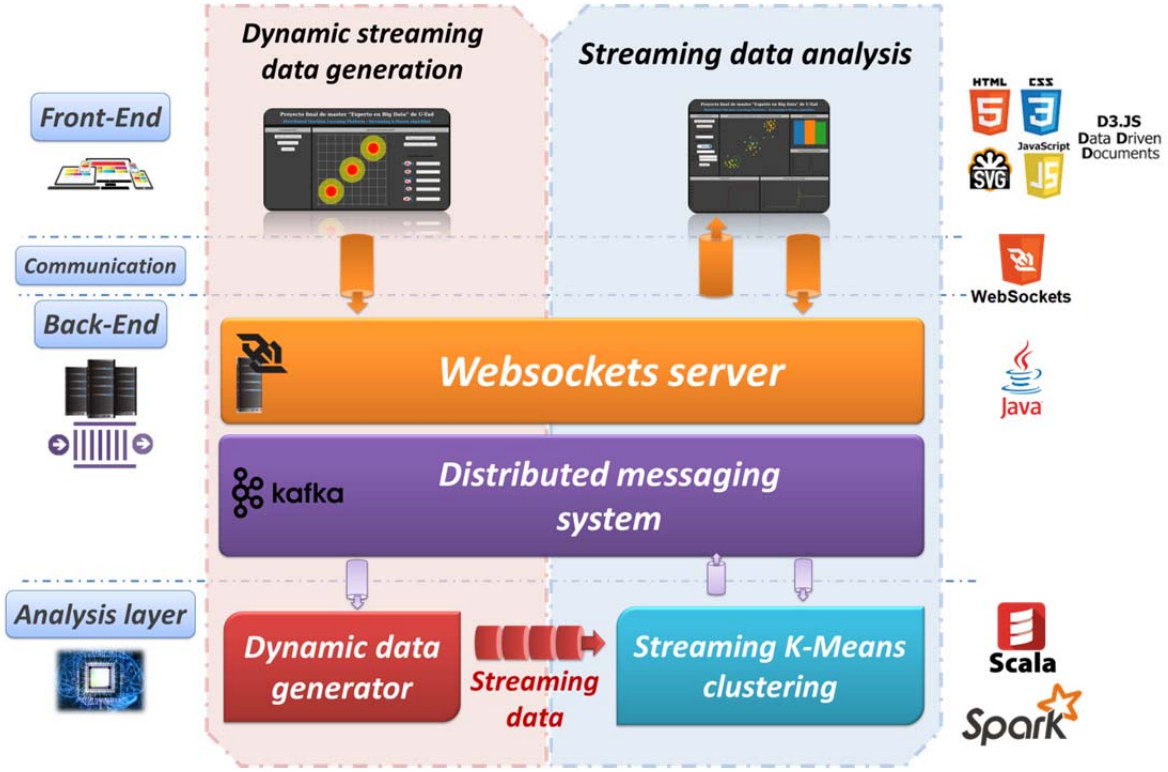


Figura 10 – Esquema con la arquitectura y tecnologías usadas.

❖ *Front-End*

Se han creado dos **interfaces web** para el **control y visualización de resultados** en el sistema de propuesto. La primera aplicación web se utiliza para controlar el modelo generativo de datos en *streaming* utilizado en el entrenamiento del modelo de clustering. La segunda aplicación, aparte de implementar un *dashboard* donde poder visualizar en tiempo real varias medidas y resultados asociados al procesado, permite variar ciertos parámetros y configuraciones de éste.

❖ *Sistema de mensajería*

Para el **intercambio de mensajes** entre la parte front y la parte de análisis se ha utilizado **Apache Kafka** [12], un sistema de mensajería basado en el modelo publicador/subcriptor, persistente, escalable, replicado, tolerante a fallos, capaz de atender cientos de megas por segundo de lecturas y escrituras provenientes de miles de clientes. Se destacan las siguientes características:

- Escrito en Scala. Creado por LinkedIn.

- Sistema **escalable** y **tolerante a fallos**.
- Se puede utilizar para servicios de mensajería (tipo ActiveMQ o RabbitMQ), procesamiento de streams, web tracking, trazas operacionales, etc.
- Categoriza los mensajes en **topics**.
- Cada uno de los nodos de kafka que forman el cluster se denomina **broker**
- **Producers**: clientes conectados a Kafka responsables de publicar los mensajes. Estos mensajes son publicados sobre uno o varios topics.
- **Consumers:** clientes conectados a Kafka subcritos a uno o varios topics responsabes de consumir los mensajes.
- Utiliza un **protocolo** propio **basado en TCP** y **Apache Zookeeper** para **almacenar el estado de los brokers**. Cada broker mantiene un conjunto de particiones (primaria y secundaria) de cada topic.
- Se pueden programar productores/consumidores en diferentes lenguajes: Java, Scala, Python, Ruby, C++ …

En este caso se ha utilizado: Apache Kafka 0.8.2.0 + Zookeeper 3.4.6.

Adicionalmente se ha utilizado Kafka-manager, una herramienta de administración de Kafka a través de una UI web. La versión utilizada ha sido la 1.2.9.10. Esta herramienta puede descargarse desde github [13].

❖ *Servidor de websockets*

Como aplicación servidor para las conexiones a través de websockets se ha utilizado el implementado en el proyecto *Kafka-Websockets*. Este proyecto implementa un servidor basado en ***Jetty*** (servidor HTTP y un contenedor de Servlets escrito en ***Java***) que actúa como proxy a través de websockets para enviar y recibir datos de Kafka desde un navegador web. Este proyecto puede ser descargado a través de github [14].

La aplicación web cliente puede consumir datos de uno o varios tópicos especificándolos en un parámetro de la *query* cuando se conecta al servidor de websokets:

/v2/broker/?topics=my_topic,my_other_topic

La aplicación cliente publica mensajes conectándose al endpoint **/v2/broker/** y mandando mensajes en formato binario o textual, debiendo incluir un campo para el tópico y otro para el mensaje; los mensajes textuales pueden incluir de forma opcional un campo key para poder realizar un mapeo entre los mensajes y las particiones en Kafka:

```
{ "topic" : "my_topic", "message" : "my amazing message" }
{ "topic" : "my_topic", "key" : "my_key123", "message" : "my amazing message" }
```

Etapa de generación de streams de datos

La **etapa de generación** de streams de datos se compone de una **aplicación web** para controlar los parámetros del modelo generativo de datos y de una **aplicación** escrita en **Scala**, encargada de leer e interpretar los **mensajes de personalización** del modelo enviados a través de Kafka por la aplicación web para generar un **stream de datos** según el modelo especificado.

❖ *Front-End – Control del modelo generativo de datos*

La aplicación web se compone de un panel de control de la conexión y envío de datos hacia la parte back-end del sistema, y de un panel para la configuración del modelo generativo de datos usado en la creación de streams de datos.

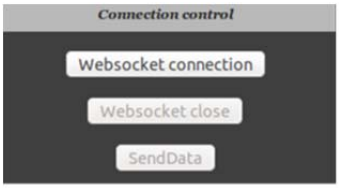


Figura 11 –Panel de control de la conexión de la aplicación web con la parte back-end del sistema

```
"Sending data:"
"{"topic":"dataGenModel","message":"[{\"mean\":[-5,-5],\"cov\":[[1,0],[0,1]]},
                                    {\"mean\":[5,5],\"cov\":[[1,0],[0,1]]},
                                    {\"mean\":[0,0],\"cov\":[[1,0],[0,1]]}]"}"
```

Figura 12 – Ejemplo de mensaje JSON que se envía desde la aplicación web.

El **panel de conexiones** (figura 11) permite abrir una conexión con el back-end a través de un websocket y así para poder enviar los mensajes de personalización del modelo generativo de datos utilizado (figura 12). En este caso se ha implementado un modelo generativo en donde cada una de las muestras se considera generadas por sólo una de entre las *K* distribuciones gaussianas multivariantes especificadas $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, escogida utilizando una distribución uniforme $\mathcal{U}(1/K)$; así, cada uno de los núcleos gaussianos se puede considerar como un cluster cuyo centroide es el parámetro $\boldsymbol{\mu}_i$.

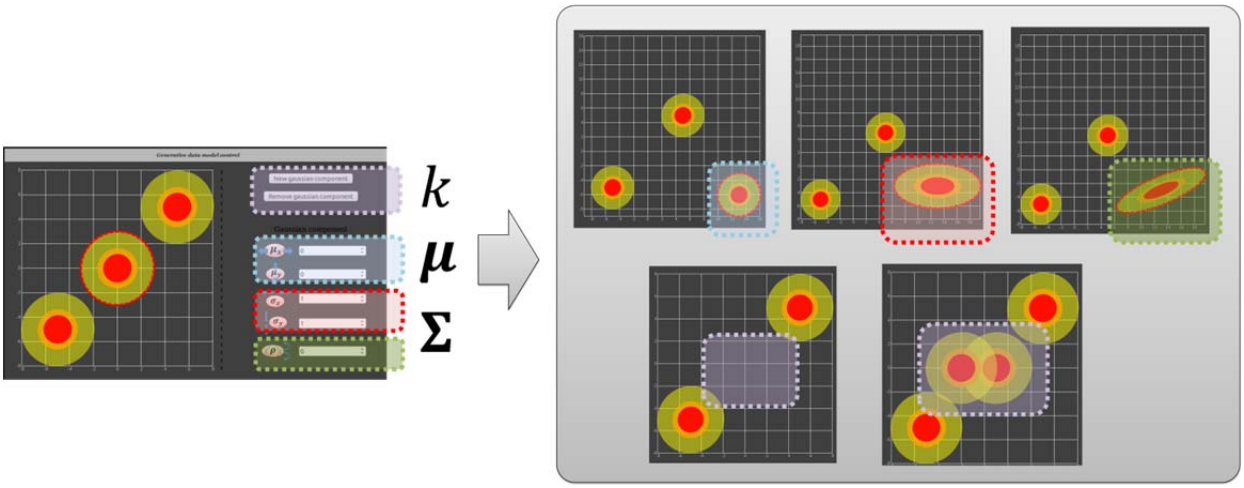


Figura 13 – Panel de control del modelo generativo de datos utilizado en el sistema.

El **panel de control del modelo generativo**, junto con el scatterplot, nos permite seleccionar una componente gaussiana y editar el valor de sus parámetros (medias y varianzas por dimensión, y el factor de correlación), así como eliminar o añadir componentes.

Se ha desarrollado una **aplicación,** utilizando el lenguaje de programación **Scala,** que ejerce como consumidor de los mensajes de configuración del modelo enviados por la aplicación web y almacenados en un tópico de Kafka. Esta aplicación se encarga de generar un streaming de datos que se ajusten al modelo que se ha configurado. Ya que el prototipo ha sido testeado en una única máquina local, se ha optado por enviar los datos al proceso de análisis a través de un socket. Para un despliegue en un cluster real, el flujo de datos en streaming lo enrutaríamos también a través de Kafka.

Etapa de análisis de streams de datos

En la etapa de análisis se ha desarrollado una **aplicación Spark** escrita en lenguaje **Scala**, encargada de la lectura de los streams de datos, el entrenamiento online del modelo de *streaming K-Means*, la obtención de estadísticas y mediciones relacionadas con el particionado de datos realizado y el envío de éstas a una aplicación web que actúa de *dashboard* de visualización de resultados.

La aplicación Spark actúa tanto como productor, como consumidor de mensajes de Kafka. El proceso driver del programa actúa como consumidor de mensajes de control para el cambio de ciertos parámetros del algoritmo y del funcionamiento del sistema de análisis; también actúa como productor para mandar los resultados y ciertas estadísticas obtenidas en el análisis (ejecutado de forma distribuida entre los workers) a la aplicación web.

Los procesos ejecutados en los workers actúan de productores, enviando un *sampling* de las muestras de entrada; en este caso, se ha tenido en cuenta las recomendaciones que aparecen en la guía de programación en Spark Streaming sobre el **patrón de diseño** a seguir cuando se deben enviar a los workers objetos de control de conexiones, como es el caso del conector con Kafka. La solución implementada está basada en la descrita en [15].

La **aplicación web**, aparte de implementar un **dashboard** de visualización de resultados, actúa como **panel de control** de diversos parámetros del sistema y del algoritmo utilizado:

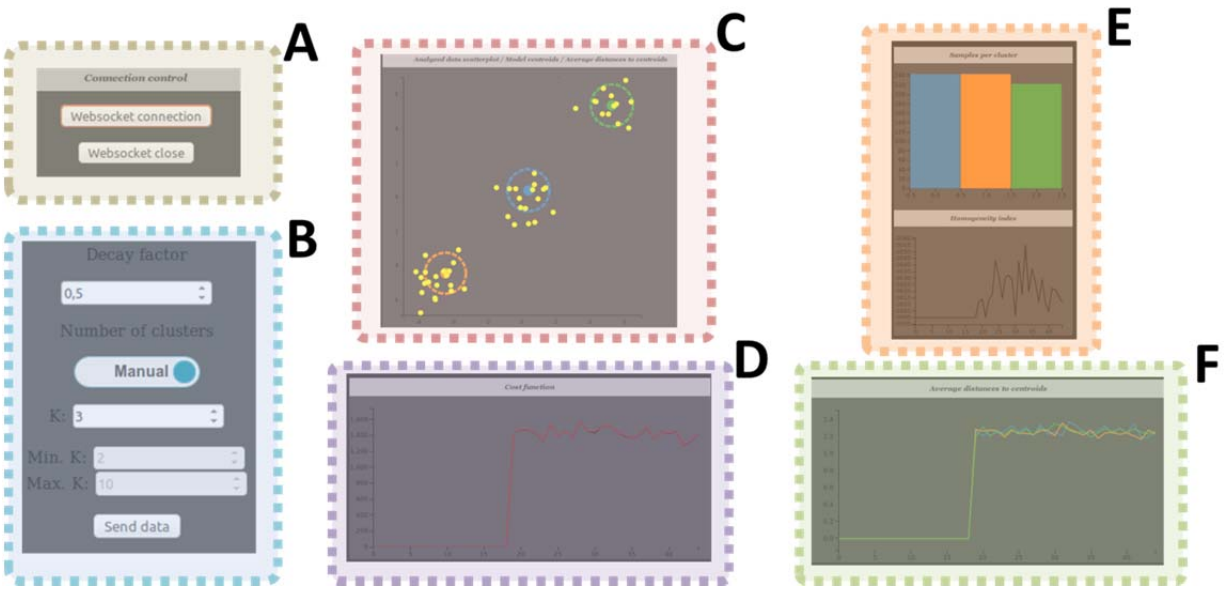


Figura 14 – Esquema de los componentes de la aplicación web desarrollada.

A. **Panel de conexión**: abre y cierra la conexión a través de websockets con el back-end.
B. **Control del sistema de procesado**: controla el valor del parámetro *decay* (factor de olvido) y el número de clusters a utilizar en el análisis: en el caso manual, el número de clusters se fija a un valor concreto, mientras que en el modo automático se fija un rango de valores posibles, siendo el proceso de análisis el encargado de entrenar un modelo por cada valor incluido en el rango, y escoger, de entre todos ellos, el que considera más adecuado.
C. **Scatterplot**: representa en “real-time” un subconjunto de los datos en stream con los que se entrena de forma dinámica el modelo de clustering, los últimos centroides asociados al modelo obtenidos, y la distancia media entre los datos de entrada y su centroide más cercano.

D. **Función de coste**: representa en “real-time” la evolución del valor de la distorsión en cada uno de los RDD/Mini-batches con los que trabaja el algoritmo. En rojo se representa el valor de la distorsión considerado óptimo (calculado a raíz del conocimiento del modelo generativo de los datos) y en negro el valor obtenido con el modelo entrenado dinámicamente.
E. **Número de muestras por cluster e índice de homogeneidad**: se representa en “real-time” el número de muestras asociadas a cada uno de los clusters para cada uno de los RDD/Mini-batches procesados. También se representa un índice heurístico que mide el grado de uniformidad conseguido; cuanto menor sea su valor, más uniforme con respecto al número de muestras asignadas a cada cluster es la segmentación conseguida.
F. **Distancias medias** de las muestras con respecto a su centroide más cercano.

Conclusiones

El objetivo del trabajo final del curso “Experto en Big Data” es el de realizar un proyecto en el que se ponga en práctica alguno o varios de los conceptos y tecnologías aprendidos en el programa, alentando a los alumnos a que, en caso de ser posible, incluyan otras tecnologías Big Data no vistas durante el curso.

Situando el presente proyecto en el contexto de los objetivos marcados en el trabajo final del curso, se puede apreciar que se han aplicado diversos conceptos introducidos en el módulo ‘**Data Analytics and Machine Learning**’ para la resolución de problemas de analítica en el marco Big Data. En este caso se ha profundizado en técnicas de segmentación o clusterización de datos, más concretamente el algoritmos basados en el modelo K-Means. El objetivo principal del proyecto es estudiar la aplicabilidad de estos algoritmos en sistemas de procesado en streaming.

Inicialmente se ha realizado un estudio teórico/práctico del método K-Means y de algoritmos utilizados en la resolución del problema que plantea este método. Situándonos en el contexto del procesado en streaming, se ha estudiado y argumentado el funcionamiento de una variante algorítmica idónea para el entrenamiento dinámico de datos ingestados en streaming. Finalmente se ha implementado un prototipo para la utilización y testeo de este algoritmo en un sistema de procesado en streaming.

Con respecto a las tecnologías utilizadas, si se analizan los trabajos prácticos realizados, se comprueba que se ha utilizado **Apache Spark** en el marco del módulo ‘**sistemas batch**’, así como **Spark Streaming** en el marco del módulo ‘**sistemas real-time**’. En la realización del proyecto se ha utilizado el lenguaje de programación **Scala**, lo cual supone la necesidad de incrementar los conceptos básicos del lenguaje que se han aprendido durante el curso.

En el prototipo propuesto se han desarrollado dos aplicaciones web que, haciendo uso y ampliando los conocimientos adquiridos en el módulo ‘**visualización**’, y utilizando **Html**, **CSS**, **Javascript**, junto con la librería **D3.js**, han permitido incluir elementos visuales interactivos que complementan al análisis Big Data en el objetivo de transformar datos en conocimiento.

Además, en el prototipo implementado, se hace uso de **Apache Kafka**, un sistema de mensajería distribuido que, por sus características de persistencia, escalado y replicación, lo convierten en una tecnología propicia para ser utilizada en entornos Big Data. Destacar también la implementación de un mecanismo de comunicación basado en **websockets**, tecnología idónea a utilizar ante la necesidad de visualización de resultados en un proceso de analítica real-time.

Bibliografía

[1] «*k*-means clustering», *Wikipedia, the free encyclopedia*.

[2] M. Inaba, N. Katoh, y H. Imai, *Applications of weighted voronoi diagrams and randomization to variance-based k-clustering (Extended Abstract)*. .

[3] S. Lloyd, «Least squares quantization in PCM», *IEEE Trans. Inf. Theory*, vol. 28, n.º 2, pp. 129-137, mar. 1982.

[4] G. Hamerly y C. Elkan, «Alternatives to the K-means Algorithm That Find Better Clusterings», en *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, New York, NY, USA, 2002, pp. 600–607.

[5] D. Arthur y S. Vassilvitskii, «K-means++: The Advantages of Careful Seeding», en *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[6] «Determining the number of clusters in a data set», *Wikipedia, the free encyclopedia*.

[7] R. Tibshirani, G. Walther, y T. Hastie, «Estimating the number of clusters in a data set via the gap statistic», *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 63, n.º 2, pp. 411-423, ene. 2001.

[8] D. T. Pham, S. S. Dimov, y C. D. Nguyen, «Selection of K in K-means clustering», *Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci.*, vol. 219, n.º 1, pp. 103-119, ene. 2005.

[9] D. Sculley, «Web-scale K-means Clustering», en *Proceedings of the 19th International Conference on World Wide Web*, New York, NY, USA, 2010, pp. 1177–1178.

[10] «Apache Spark™ - Lightning-Fast Cluster Computing». [En línea]. Disponible en: http://spark.apache.org/.

[11] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, y S. Vassilvitskii, «Scalable K-means++», *Proc VLDB Endow*, vol. 5, n.º 7, pp. 622–633, mar. 2012.

[12] «Apache Kafka». [En línea]. Disponible en: http://kafka.apache.org/.

[13] «yahoo/kafka-manager», *GitHub*. [En línea]. Disponible en: https://github.com/yahoo/kafka-manager.

[14] «b/kafka-websocket», *GitHub*. [En línea]. Disponible en: https://github.com/b/kafka-websocket.

[15] «Spark and Kafka integration patterns», *allegro.tech blog*. [En línea]. Disponible en: http://allegro.tech/2015/08/spark-kafka-integration.html.