Universidad Nacional de Educación a Distancia

Tesis de Máster

Arquitectura lambda aplicada a clustering de documentos en contextos Big Data

Máster universitario en I.A. avanzada:
Fundamentos, métodos y aplicaciones

Autor:

Alberto Vallejo Martínez

Directores:

Raquel Martínez Unanue

Álvaro Rodrigo Yuste

Octubre de 2015

Contenido

Lista de Figuras

Lista de Tablas

Resumen

La aplicación de técnicas de minería de datos, concretamente de clustering, sobre grandes volúmenes de datos (Big Data) supone un desafío en cuanto a la escalabilidad y al tiempo de respuesta, ya que cantidades crecientes de datos implican tiempos mayores de computación.

La arquitectura lambda es un conjunto de recomendaciones de propósito general para diseñar una arquitectura en escenarios Big Data de forma que se resuelva el problema de la latencia y se puedan obtener resultados en tiempo real.

En este trabajo se presenta un estudio aplicando la arquitectura lambda sobre el clustering de documentos en contextos Big Data. La problemática que se desea resolver es la elevada latencia que tiene lugar cuando se introducen nuevos documentos en el sistema que realiza clustering. Uno de los puntos clave sugeridos por esta arquitectura es la separación del procesamiento en tres capas: batch layer, speed layer y serving layer.

Una problemática adicional al tratar documentos es su elevada dimensionalidad y este problema se soslaya mediante reducción de dimensionalidad con Latent Dirichlet Allocation.

Los experimentos se han llevado a cabo utilizando el framework Apache Spark y demuestran que esta combinación de capas permite realizar clustering sobre grandes volúmenes de datos y disponer de resultados actualizados en tiempo real, con calidad del clustering comparable a trabajos similares sobre contextos no Big Data.

Palabras clave

Arquitectura lambda, clustering de documentos, Big Data, LDA, streaming, Apache Spark, k-means

Abstract

The application of data mining techniques, specifically clustering on large volumes of data (such as Big Data), represents a challenge in terms of scalability and response time, given that a higher amount of data involves a higher computation time.

The lambda architecture offers a set of general purpose recommendations to design an architecture in Big Data scenarios with the purpose of reducing latency and getting results in real time.

This paper presents a study on applying the Lambda architecture on the clustering of documents in a Big Data environment.

We focus on the problem of the high latency that occurs when new documents are ingested into the clustering system.

One of the key points suggested by this architecture is a separation into three layers of processing: batch layer, serving layer and speed layer.

An additional problem in dealing with documents is the high dimensionality when representing such documents, which is approached by means of dimensionality reduction using Latent Dirichlet Allocation.

We have seen that the combination of layers proposed in the Lambda Architecture allows clustering of large data volumes and yields results updated in real-time, obtaining a clustering quality comparable to other approaches that do not work on Big Data.

Keywords

Lambda architecture, document clustering, Big Data, LDA, streaming, Apache Spark, k-means

Los metafísicos de Tlön no buscan la verdad ni siquiera la verosimilitud: buscan el asombro.

Jorge Luis Borges, "Tlön, Uqbar, Orbis Tertius"

1 Introducción

1.1 Motivación

Nos encontramos inmersos ante un fenómeno de explosión de la información disponible en formato digital, y parece que la tendencia va a seguir en el futuro hacia un crecimiento de los datos disponibles. En 2010 las empresas y los usuarios almacenaron más de 13 exabytes de nuevos datos (Labrinidis & Jagadish, 2012). De toda esta masiva cantidad de datos que se ha denominado Big Data (Diebold, 2003), puede extraerse información relevante mediante técnicas de minería de datos y aprendizaje automático. Estas técnicas pueden usarse para detectar patrones en este océano de datos que se produce cada segundo. Con este fin, adquiere importancia la investigación de técnicas para la generación, procesamiento e interpretación de esta creciente cantidad de datos disponibles.

Centrándonos en concreto en la información textual disponible en internet, un 80% de la información en internet está almacenada en forma de texto (Xiao, 2010). En este ámbito son de aplicación técnicas de la minería de textos como la clasificación automática y el clustering de textos.

Un área de interés en particular es el clustering de documentos, que consiste en agrupar los documentos en grupos de forma que los objetos en cada grupo compartan entre sí más similitud que frente a otros objetos de otros grupos (Xu & Wunch, 2009). Esta tarea constituye una cuestión primordial en el área de Recuperación de Información (IR, Information Retrieval) y encuentra aplicaciones en el mundo real como por ejemplo en motores de búsqueda (Baeza-Yates & Ribeiro-Neto, 1999).

Aplicar técnicas de clustering en Big Data es uno de los nuevos desafíos pendientes, ya que los algoritmos de clustering presentan elevados costes computacionales y en Big Data tienen que enfrentarse a volúmenes de datos enormes. La cuestión es como hacer frente a este problema y obtener los resultados en un tiempo razonable (Shirkhorshidi et al 2014).

1.2 Descripción del problema

Para hacer frente a este crecimiento de datos, han surgido una serie de tecnologías que permiten realizar cálculos y escalar a conjuntos de datos extraordinariamente grandes. Muchas de estas técnicas fueron desarrolladas inicialmente por Google, incluyendo sistemas de ficheros distribuidos como Google File System (Ghemawat et al., 2003), computación en paralelo a gran escala mediante Map-Reduce (Dean & Ghemawat, 2004) y BigTable (Chang et al., 2006). La comunidad open source reaccionó en los años siguientes desarrollando numerosos proyectos como Hadoop[1], HBase[2], MongoDB[3], Cassandra[4], RabbitMQ[5], etc.

[1] https://hadoop.apache.org/
[2] http://hbase.apache.org/
[3] https://www.mongodb.org/
[4] http://cassandra.apache.org/
[5] https://www.rabbitmq.com/

La parte clave de estos sistemas es que pueden escalar a conjuntos de datos enormemente más grandes comparado con el tratamiento tradicional de los datos que se realizaba mediante bases de datos relacionales (Strauch, 2011).

Son múltiples en la bibliografía los trabajos que usan Hadoop para resolver cálculos sobre volúmenes de datos muy grandes aprovechando su capacidad para realizar ejecuciones de trabajos por lotes en paralelo (Dittrich & Quian, 2012).

Sin embargo el framework Map-Reduce está más adaptado a ejecutar procesamiento por lotes sobre grandes volúmenes de datos (Condie et al., 2010), lo que puede ser un inconveniente al manejar procesamiento en tiempo real.

El problema que se desea resolver tiene lugar en un escenario Big Data en el cual se añaden nuevos documentos, ya que si se desean obtener resultados actualizados, el modelo completo tiene que ser reentrenado (Zhao et al., 2014).

Para afrontar los desafíos de Big Data surgen nuevos enfoques que se replantean la arquitectura de los sistemas desde cero. La aproximación que se estudia en este trabajo es la Arquitectura Lambda propuesta por (Marz & Warren, 2015). Esta arquitectura tiene la ventaja de presentar una baja latencia frente a la incorporación de datos nuevos (Singh, 2014). Se trata de una arquitectura de carácter general, y se ha aplicado con éxito a problemas concretos que requieren baja latencia al incorporar nuevos datos.

En este trabajo se estudia la aplicación de la arquitectura lambda, concretamente al área de clustering de documentos sobre colecciones de datos masivas que además crecen en el tiempo, como posible respuesta al problema de la elevada latencia que se observa en el procesamiento de nuevos documentos. Considero que la arquitectura lambda puede ser aplicable al clustering de documentos porque se trata de una arquitectura de carácter general y puede aplicarse a distintos tipos de problemas que trabajen con Big Data.

El método diseñado, por tanto, deberá ser capaz de realizar computaciones de clustering sobre colecciones masivas de datos históricos, pero también deberá ser capaz de realizar cálculos sobre nuevos documentos que se incorporan y devolver los resultados en tiempo real.

Con la intención de responder a la incorporación de nuevos documentos y aplicando la arquitectura lambda, se utilizarán técnicas que trabajan sobre flujos de datos o streaming aplicadas al clustering.

Adicionalmente, la estrategia utilizada en este trabajo para realizar el clustering y resolver el problema presente en Big Data del volumen de datos, consistirá en combinar por una parte una reducción de dimensionalidad y por otra la ejecución del algoritmo de forma paralela/distribuida.

No se ha encontrado en la bibliografía trabajos que apliquen la arquitectura lambda al clustering de documentos. Además, el interés de avanzar en el estado del arte sobre el área del clustering en tiempo real en contextos Big Data surge de la necesidad de capturar el dinamismo presente en grandes colecciones de documentos que evolucionan en el tiempo. Un ejemplo de ello lo encontramos en buscadores de noticias en internet donde la temática y las relaciones entre los términos y los documentos varían en función de los temas de actualidad y es importante la inmediatez de la actualización de esas relaciones en el modelo.

1.3 Objetivos

Los objetivos principales de este trabajo de investigación son los que a continuación se enumeran:

- Hacer un estudio sobre los trabajos actuales que utilicen la arquitectura lambda.
- Evaluar el estado del arte sobre los trabajos actuales en cuanto a clustering, reducción de la dimensionalidad y procesamiento de flujos de datos en contextos Big Data.
- Realizar una propuesta diseñada, desde un punto de vista arquitectural, según las directivas de la arquitectura lambda. En esta propuesta se utilizarán una combinación de estrategias básicas para abordar el problema de la escalabilidad en el clustering de documentos sobre contextos Big Data.
- Obtener una serie de medidas de escalabilidad y calidad del clustering que permitan comparar los resultados obtenidos con otros trabajos y que se puedan valorar las decisiones tomadas.

La arquitectura lambda expone unas directrices en cuanto a la organización en capas para conseguir unas propiedades deseables en los sistemas Big Data, y en cada capa es posible utilizar una serie de tecnologías. La construcción de la propuesta implicará el diseño de estas capas.

La importancia de alcanzar los objetivos en este estudio radica en que permitirá verificar que la arquitectura lambda aplicada al clustering de documentos sobre contextos Big Data con incorporación de nuevos documentos, proporciona resultados en tiempo real y con una calidad del clustering comparable a otros trabajos similares. Para ello será necesario lidiar con problemas inherentes a los documentos como su elevada dimensionalidad.

1.4 Estructura de la memoria

En el capítulo actual de "Introducción" (página 10) se han expuesto los objetivos del trabajo y se ha introducido la problemática asociada.

En el siguiente capítulo "Preliminares" (página 13) se exponen las distintas áreas que están relacionadas en este trabajo (Big Data, arquitectura lambda, clustering, reducción de dimensionalidad, streaming). Además, se detallan sistemas similares que realizan parte de la funcionalidad de cada área, y se exponen las técnicas empleadas.

En el capítulo "Diseño del método propuesto" (página 40) se detalla un método que aplica la arquitectura lambda concretamente al clustering de documentos en contextos de datos masivos.

En el capítulo "Experimentos y resultados" (página 49) se describe el diseño de los experimentos, los resultados obtenidos y un análisis sobre los mismos.

Finalmente, las principales conclusiones y las líneas futuras que quedan abiertas a la investigación se recogen en el capítulo "Conclusiones y trabajos futuros" (página 64).

2 Preliminares

Este trabajo es transversal en el sentido de que está relacionado con diferentes áreas como el clustering, procesamiento de flujos de datos (streaming), reducción de dimensionalidad, o Big Data. En esta sección se describen someramente los principales conceptos y algoritmos de cada una de ellas, o en su caso se explica la problemática que ha llevado a la necesidad de utilizar cada uno de ellos, y el estado del arte de los trabajos de cada área.

2.1 Big Data

2.1.1 Generalidades

Aunque no hay consenso en la definición del término Big Data, está ampliamente aceptado decir que Big Data es "cuando el tamaño de los datos en sí mismo es parte del problema" (Loukides 2010).

Es común en la bibliografía referirse a que Big Data se caracteriza por las tres V (Zikopoulos, 2012)(Laney, 2001):

- Volumen
- Variedad
- Velocidad

Volumen: es el atributo que principalmente define a Big Data y se refiere a la habilidad de poder manejar grandes cantidades de datos. En este sentido hay que tener en cuenta que los criterios que se consideran son: el tamaño del conjunto de datos, la elevada dimensionalidad y el manejo de los datos ruidosos.

Variedad: Se refiere a la capacidad del sistema para manejar datos de diferentes fuentes y diferentes tipos de datos (numéricos, categorías y jerarquías). En este aspecto influye el tipo de conjunto de datos y las características de los datos.

Velocidad: Se refiere a la frecuencia de generación de los datos o de entrega de los datos. Por ejemplo, el flujo de datos procedente de páginas en internet o tuits en twitter.

2.1.2 Desafíos de Big Data

El principal desafío para las aplicaciones Big Data es explorar grandes volúmenes de datos y extraer información útil o conocimiento para acciones futuras (Rajaraman & Ullman, 2012). A pesar de que las posibilidades que permite Big Data son reales, todavía hay una amplia grieta entre su potencial y su realización (Labrinidis & Jagadish, 2012).

Los problemas con Big Data de heterogeneidad, escalabilidad, puntualidad de los datos, complejidad y privacidad impiden el progreso en todas las fases del procesamiento que pueden crear valor a partir de los datos.

Los problemas se aprecian durante la adquisición de datos, sobre qué datos mantener y cuáles descartar, y la forma de almacenarlos de forma fiable junto con los metadatos correctos.

El valor de los datos explota cuando se pueden vincular con otros datos, por tanto, la integración de los datos es un importante creador de valor. Como la mayoría de los datos se generan directamente en formato digital, se tiene la oportunidad de vincular automáticamente los datos creados anteriormente mediante técnicas de minería de datos.

El análisis de datos, la organización, la recuperación y el modelado son otros retos fundamentales. El análisis de datos es un claro cuello de botella en muchas aplicaciones, tanto debido a la falta de escalabilidad de los algoritmos subyacentes como debido a la complejidad de los datos que necesitan analizarse. Por último, la presentación de los resultados y su interpretación por expertos en dominios no técnicos es crucial para la extracción del conocimiento (Labrinidis & Jagadish, 2012).

En el procesamiento en Big Data se encuentran las siguientes fases (Labrinidis & Jagadish, 2012):

- Adquisición, registro
- Extracción, limpieza, anotación
- Integración, agregación y representación
- Análisis y modelado
- Interpretación

Supone un desafío para Big Data la satisfacción de las siguientes necesidades:

- Heterogeneidad
- Escalabilidad
- Prontitud
- Privacidad
- Colaboración humana

Las fases y las necesidades enumeradas se muestran en la Figura 2.1.

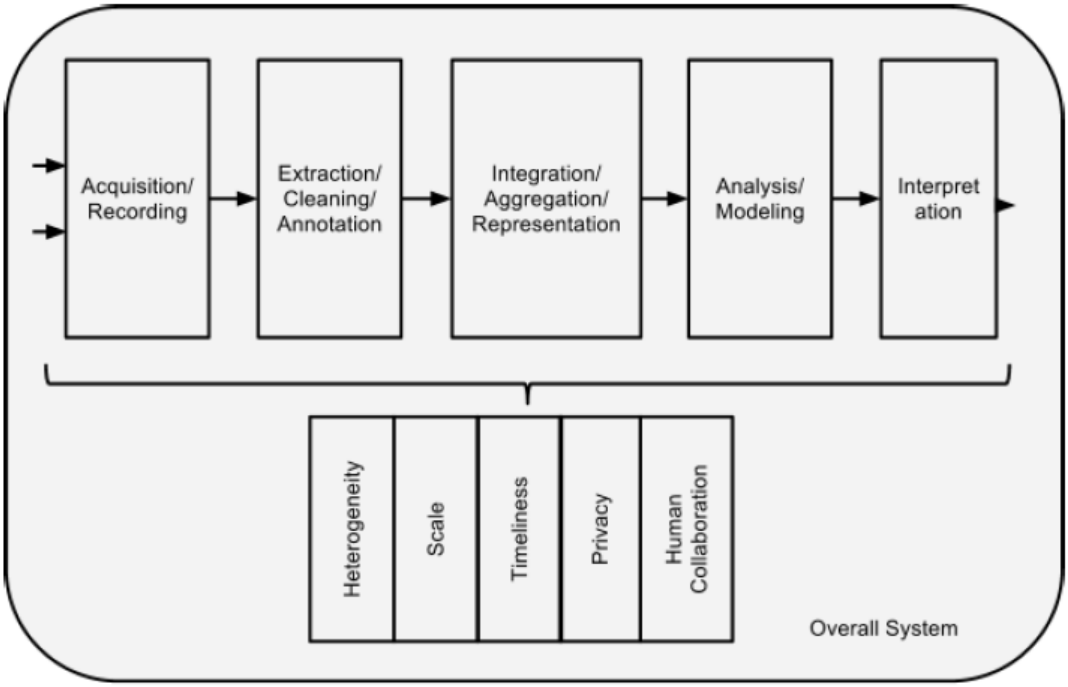

Figura 2.1. Fases del procesamiento en Big Data. (Fuente Labrinidis & Jagadish, 2012)

Este trabajo se centra en las fases de integración, agregación y representación junto con análisis y modelado, aunque en cierta medida también se toman decisiones relacionadas con otras fases como registro y limpieza. En cuanto a las necesidades, este trabajo se centra en la escalabilidad y la prontitud.

La gestión de grandes volúmenes de datos y en rápido aumento ha sido un tema difícil durante décadas para conseguir escalabilidad. En el pasado, este desafío fue mitigado con procesadores cada vez más rápidos (aprovechando la Ley de Moore). La introducción del framework Map-Reduce ha permitido ejecutar de forma distribuida trabajos de computación sobre clusters de máquinas.

Para que los usuarios puedan construir complejos procesamientos de datos sobre Big Data es esencial que dispongan de primitivas adecuadas de alto nivel para especificar sus necesidades en este tipo de sistemas. El framework Map-Reduce, el cual se analiza en la siguiente sección, ha sido de tremendo valor, pero es un primer paso y es deseable disponer de herramientas de más alto nivel, especificaciones declarativas para cada fase implicada en el procesamiento de Big Data (Labrinidis & Jagadish, 2012).

2.1.3 Map-Reduce

Map-Reduce es un modelo de programación para dar soporte a la computación paralela sobre grandes colecciones de datos en clusters de computadoras. La idea principal de Map-Reduce es ocultar al usuario los detalles de la ejecución paralela y permitirle enfocarse solamente en las estrategias de procesamiento de datos (Dean & Ghemawat, 2004).

El modelo consiste en dos funciones primitivas "Map" y "Reduce" (Figura 2.2). La entrada de "Map" es un conjunto de pares clave-valor (k_1 v_1) a los cuales se les aplica una función "Map" que devuelve como resultado un conjunto intermedio de pares clave-valor (k_2 v_2). Este conjunto intermedio se agrupa según claves iguales, las cuales sirven de entrada para la función "Reduce", la cual trabaja sobre toda la lista de valores asociados a la misma clave y produce cero o más resultados agregados en forma de lista (list v_3). Los conjuntos de pares clave-valor pueden pertenecer a dominios diferentes.

Map

$Map(k_1, v_1) \rightarrow list(k_2, v_2)$

Reduce

$Reduce(k_2, list\ (v_2)) \rightarrow list(v_3)$

Figura 2.2. Signatura de funciones Map y Reduce

Existen varias implementaciones de Map-Reduce, por ejemplo Apache Hadoop[6] es una implementación open source en Java. En la sección 2.1.4 se comparará frente a Apache

[6] http://hadoop.apache.org/

Spark, que es un framework para trabajar en Big Data que se ha utilizado en los experimentos de este trabajo.

2.1.4 Apache Spark

Apache Hadoop es un framework open source que ofrece soluciones en Big Data para procesar grandes conjuntos de datos en clusters de computadoras usando modelos de programación sencillos. Hadoop se compone principalmente de dos componentes: HDFS (Hadoop Distributed File System) (Shvachko et al., 2010) y el framework de Map-Reduce (Dean & Ghemawat, 2004).

Hadoop está diseñado para ser simple y fácil de usar, flexible en cuanto al procesamiento y almacenamiento de datos, tolerante a fallos y con elevada escalabilidad.
No obstante, presenta una serie de inconvenientes (Gopalani & Arora, 2015):

- Flujo de datos fijo: Provee facilidad de uso con una abstracción simple que también es un flujo de datos fijo. Es decir, varios algoritmos complejos son difíciles de implementar en un solo trabajo de procesamiento (Job). Además algunos algoritmos que requieren múltiples entradas no son compatibles.
- Baja eficiencia: Con la tolerancia a fallos y la escalabilidad como objetivos primarios, las operaciones no siempre son las más eficientes.
- Latencia: Por su inherente procesamiento en batch, sufre el problema de la latencia.

Sin embargo recientemente, con la introducción de Apache Spark[7], se dispone de un nuevo modelo de computación en el contexto de Big Data que ofrece una interfaz de programación que permite disminuir los esfuerzos de programación y ofrece mejor rendimiento en los tipos de problemas relacionados con Big Data (Gopalani & Arora, 2015).

Mientras que Hadoop ofrece poca flexibilidad a la hora de crear flujos de datos ya que sigue un esquema de ejecución fijo, Spark ofrece un esquema de computación más flexible, debido a que se basa en un flujo de ejecución de grafo acíclico dirigido, y es posible modificar el flujo con distintas transformaciones y acciones (Gopalani & Arora, 2015).

Spark es un motor de computación en cluster, rápido y de propósito general para procesamiento de datos a gran escala. Comenzó como un proyecto de investigación en UC Berkeley en el AMPLab, con el objetivo de diseñar un modelo de programación compatible con una clase de aplicaciones más amplia que Map-Reduce mientras mantenían la tolerancia a fallos.

Además Spark dispone de una librería de aprendizaje automático (MLlib), consulta de datos con SQL (Spark SQL) y procesamiento de flujos (Spark Streaming).

Spark expone un interfaz de programación funcional (Zaharia et al., 2010) y puede utilizarse con varios lenguajes de programación (Java, Scala, Python, R). La principal fortaleza de la programación funcional, al evitar el estado (y los efectos colaterales), es que el sistema entero obtiene transparencia referencial, que implica que cuando se pasan una serie de argumentos a una función siempre devuelve el mismo resultado, es decir, siempre se comporta de la misma forma, lo cual facilita su ejecución distribuida (Moseley & Marks, 2006).

[7] https://spark.apache.org/

En la Figura 2.3 se esquematiza los módulos disponibles en Spark:

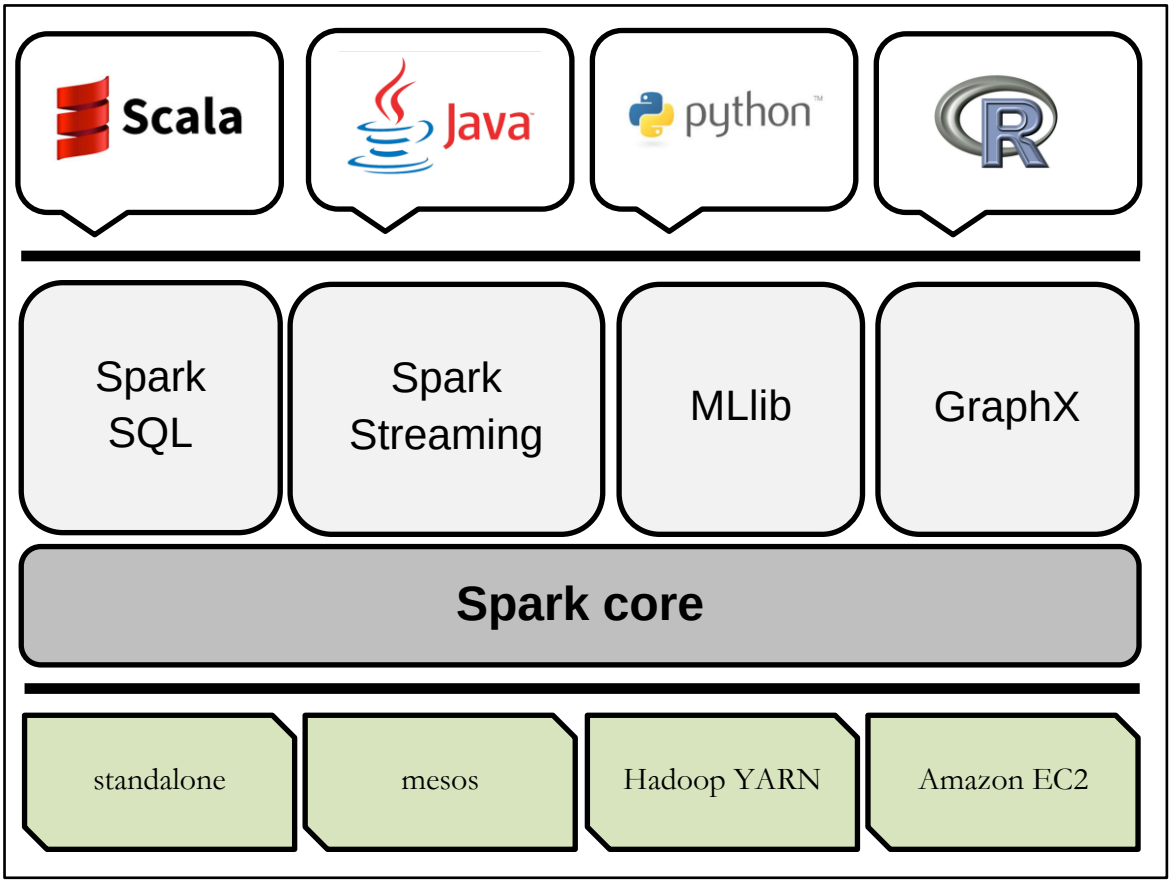

Figura 2.3. Arquitectura de Apache Spark

Spark ofrece una abstracción llamada Resilient Distributed Datasets (RDD) para soportar las aplicaciones que trabajan con datos masivos de forma eficiente. Los datos en RDD se pueden almacenar en memoria entre consultas sin necesidad de replicación lo cual ofrece un mejor rendimiento. Los RDD permiten mejorar los modelos existentes hasta 100 veces en análisis que requieran procesamiento iterativo. También permiten minería de datos interactiva, consultas SQL altamente eficientes, procesamiento de flujos y computación en paralelo sobre grafos (Zaharia, 2010) (Zaharia, 2012) (Da Silva, 2015).

En la Figura 2.4 se muestra una comparativa de tiempos de respuesta para comparar Spark y Hadoop (Zaharia et al., 2012). En la evaluación se realizaron cálculos de regresión logística sobre 100 GB de datos en varias iteraciones y se observa que mientras que Hadoop dedica un tiempo constante por iteración de 110 segundos, Spark tarda 80 segundos en cargar los datos en memoria en la primera iteración, pero solo seis segundos para cada subsiguiente iteración.

Asimismo Spark dispone de otra abstracción para manejar los flujos llamada D-Stream (Discretized Stream) que se basa en tratar los cálculos de flujos como series de computaciones determinísticas en lotes sobre pequeños intervalos de tiempo (Zaharia, 2012).

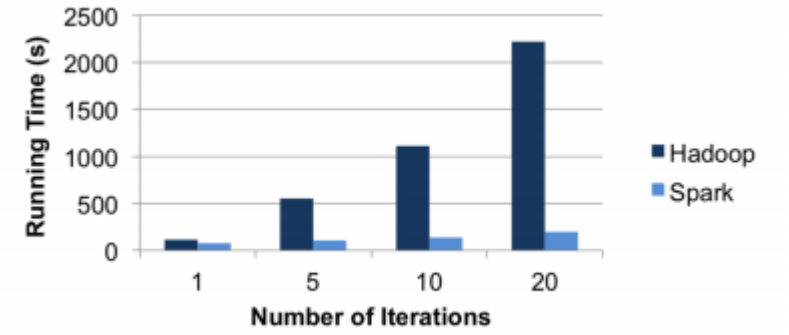

Figura 2.4. Comparativa de rendimiento Hadoop vs Spark, ejecutando Regresión Logística. (Fuente: Zaharia et al., 2012)

2.2 Arquitectura lambda

Los autores de la Arquitectura Lambda (Marz & Warren, 2015) la definen como un conjunto de principios para arquitecturar sistemas Big Data en tiempo real.

En un sistema Big Data son deseables las siguientes propiedades: (Kumar et al., 2014)

- Robustez y tolerancia a fallos
- Escalabilidad
- Generalización
- Extensibilidad
- Consultas a medida
- Mantenimiento mínimo
- Depurabilidad
- Actualizaciones con baja latencia

El principal objetivo que intenta conseguir la arquitectura lambda es construir los sistemas Big Data de forma que satisfagan todas estas propiedades (Marz & Warren, 2015). Para ello se organiza en una serie de capas (Figura 2.5): capa por lotes (en adelante capa Batch), capa de velocidad (en adelante capa Speed) y capa proveedora (en adelante capa Serving) que describiremos más adelante. Cada capa satisface un subconjunto de estas propiedades.

Conjuntamente, la capa batch y la capa serving satisfacen casi todas las propiedades, excepto la baja latencia de actualización de nuevos datos. Esto es, soportan consultas arbitrarias sobre un juego de datos arbitrario, con la desventaja de que esas consultas estarán desactualizadas en unas pocas horas. Cuando llegan nuevos datos, los cálculos necesitan unas pocas horas para propagarse a través de la capa batch hasta la capa serving donde pueden consultarse. En sistemas donde es importante que exista una baja latencia en la actualización de nuevos datos es necesario introducir una nueva capa, la capa speed, que proporciona esta propiedad. Mediante esta arquitectura, colectivamente las tres capas cumplen todas las propiedades (Marz & Warren, 2015).

(Marz & Warren, 2015) también sugieren buenas prácticas para almacenar los datos en un conjunto de datos maestro (master dataset) que es inmutable, donde solo se añaden datos sin ningún procesamiento ('crudos') y a partir de los cuales es posible regenerar todo el modelo.

Usando un esquema de datos inmutables se obtienen dos ventajas:

- Tolerancia al fallo humano.
- Simplicidad y flexibilidad a la hora de incorporar cambios al sistema.

Otro punto a destacar de esta arquitectura es que se consigue reducir la complejidad porque evita la necesidad de escrituras aleatorias en el sistema de base de datos y esto hace innecesarios los mecanismos de bloqueo.

Tal y como hemos adelantado, la arquitectura lambda propone una división en tres capas que a continuación analizamos:

- Capa por Lotes (Batch Layer)
- Capa de Velocidad (Speed Layer)
- Capa Proveedora (Serving Layer)

Capa por Lotes (Batch Layer)

Se encarga de:

- Almacenar el conjunto de datos maestro que es inmutable y crece constantemente.
- Crear vistas desde este conjunto de datos.

Esta computación se ejecuta en batch, se repite continuamente en iteraciones y conforme llegan nuevos datos no se procesan inmediatamente, sino que se encolan en el conjunto de datos maestro para agregarse a las vistas en la siguiente iteración.

Capa de Velocidad (Speed Layer)

El resultado de la ejecución de la capa batch no satisface el requisito de tiempo real ya que su ejecución puede llevar tiempo en la creación y propagación de las vistas. Esta capa de velocidad compensa la carencia de la capa batch y permite disponer de resultados actualizados.

Esta capa es responsable de:

- Calcular las vistas cuando llegan nuevos datos.
- Intentar resolver la elevada latencia generando vistas en tiempo real con los incrementos de nuevos datos.
- Esta capa típicamente utiliza algoritmos incrementales.

En esta capa se puede asumir la obtención de una calidad inferior en los resultados, ya que finalmente será corregida en el próximo ciclo de ejecución de la capa batch. La capa speed almacena considerablemente menos datos que la capa serving. La separación de roles y responsabilidades limita la complejidad de la capa speed.

Capa Proveedora (Serving Layer)

Es responsable de servir los datos que proceden de las dos capas anteriores: por una parte los resultados de la computación en batch y por otra los incrementos procedentes de datos nuevos. Por tanto se encarga de indexar y exponer las vistas (de sólo lectura) para que puedan ser consultadas. Las consultas se realizan bajo demanda.

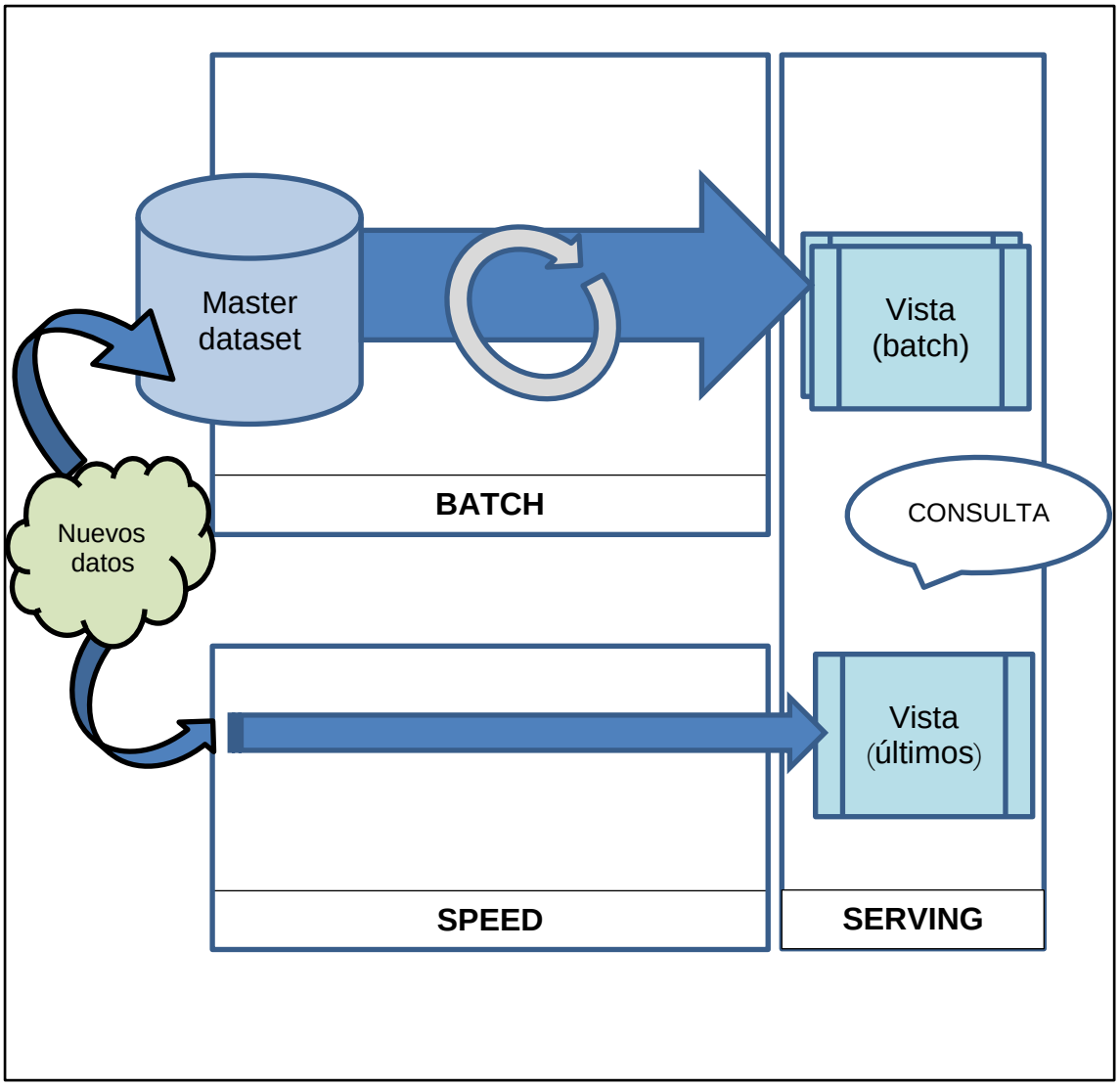

Figura 2.5. Diagrama con las capas de la arquitectura lambda.

Típicamente cada capa se puede implementar mediante las tecnologías indicadas en la Tabla 2.1.

Capa	Tecnologías
Capa por lotes/Batch Layer	Apache Spark Core Apache Hadoop (Map-Reduce)
Capa de velocidad/Speed Layer	Apache Spark Streaming Apache Storm Apache Samza Moa
Capa Proveedora/Serving Layer	Apache Hive ElephanDB Druid Cloudera Impala Apache Cassandra

Tabla 2.1. Ejemplo de tecnologías aplicables a cada capa de la arquitectura lambda

En este trabajo se ha decidido utilizar Apache Spark, que ya se analizó en la sección 2.1.4, para evitar construir un sistema con tecnologías heterogéneas, que implicaría utilizar más APIs de programación y más complejidad a la hora de integrar los datos entre capas.

La capa batch trabaja sobre datos históricos mientras que la capa speed trabaja sobre los datos nuevos.

Para ilustrar el procesamiento de los datos, seguidamente se muestra un ejemplo. En la Figura 2.6 se muestra el inicio del procesamiento de la capa batch sobre el juego de datos completo. La capa batch puede requerir un tiempo prolongado de proceso, durante el cual pueden llegar datos nuevos (Figura 2.7), cada uno de ellos se procesan inmediatamente en la capa speed y para cada uno se rinden resultados inmediatamente. Simultáneamente, estos mismos nuevos datos se encolan para agregarse al juego de datos maestro de forma que estén disponibles para nuevos ciclos de procesamiento de la capa batch. Cuando la capa batch termina su procesamiento (Figura 2.8) es cuando rinde los resultados de esta capa. Los nuevos datos que se han agregado durante este ciclo se añaden al juego de datos maestro. En este punto vuelve a empezar la ejecución de un nuevo ciclo de la capa batch sobre la colección completa. Los datos que han ido generando ambas capas están disponibles para su consulta bajo demanda desde la capa serving, que permite hacer consultas tanto sobre datos históricos como sobre los datos nuevos, con la ventaja de que los cálculos sobre los datos nuevos están disponibles inmediatamente.

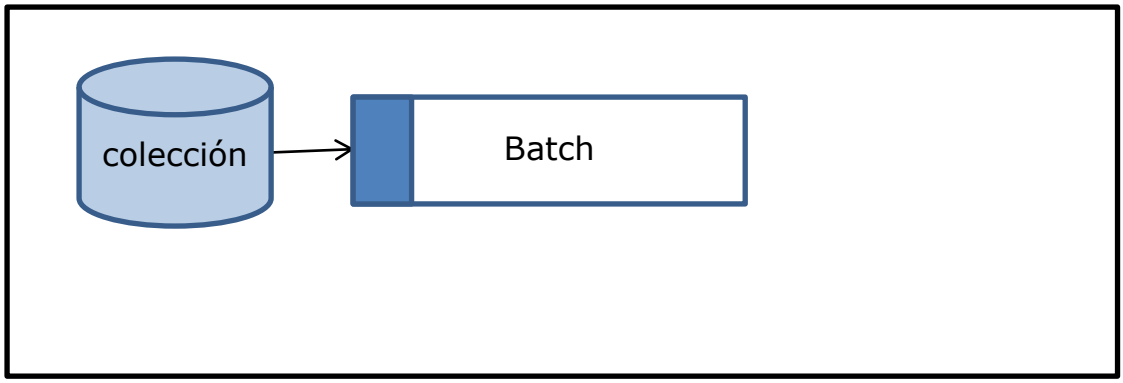

Figura 2.6. Ejemplo. Se inicia el procesamiento de la capa batch sobre el dataset completo

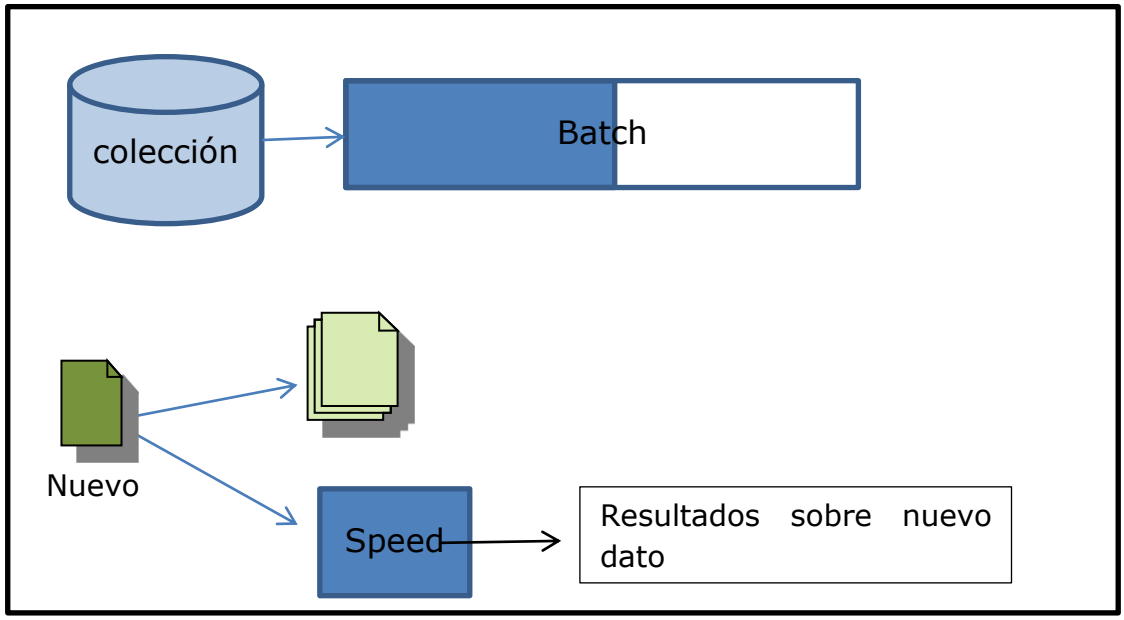

Figura 2.7. Ejemplo. Llega un documento nuevo

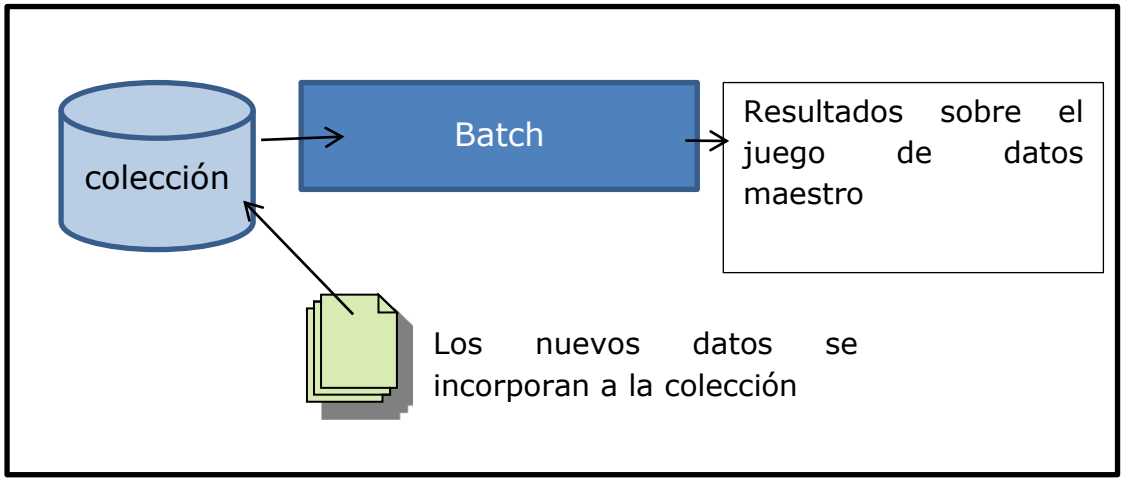

Figura 2.8. Ejemplo. La capa batch termina su procesamiento

2.2.1 Ventajas y aplicaciones de la arquitectura lambda

Existen numerosos trabajos (Liu et al., 2014) (Fan & Bifet, 2013) (Singh, 2014) (Chen et al., 2014) que destacan como ventaja de la arquitectura lambda la capacidad de trabajar con datos masivos y obtener respuestas en tiempo real. Ello permite trabajar paralelamente con datos históricos y en tiempo real.

La arquitectura lambda es versátil, se trata de una técnica de uso general que se puede adoptar en diversas aplicaciones: sistemas de gestión financiera, redes sociales, aplicaciones, científicas, analíticas de medios sociales (Kumar et al., 2014). (Singh, 2014) indica que una línea abierta para futuros trabajos es la aplicación de esta arquitectura destinada al aprendizaje supervisado y no supervisado.

Otros autores (Kitchin 2014) (Kumar et al., 2014) se detienen en señalar la ventaja de la robustez de esta arquitectura para reducir potenciales errores y conseguir que la aplicación sea más robusta.

En (Shih et al., 2014) realizan un estudio aplicado al procesamiento de eventos en tiempo real procedentes de redes de telefonía. El sistema permite introducir reglas basadas en los resultados analíticos para disparar campañas de marketing. Su sistema se basa en la arquitectura lambda e implementa la capa speed mediante Apache Storm. La ventaja que aporta la aplicación de la arquitectura lambda, es que permite realizar el proceso gestionando millardos de métricas en tiempo real comparado con sistemas tradicionales basados en almacenes de datos tradicionales que realizan el procesamiento de este volumen de datos por lotes pero no en tiempo real.

En (Cortés et al., 2014) se realiza un trabajo sobre la explotación de la información climática, así como del consumo energético doméstico, para crear modelos de predicción basados en aprendizaje automático, del consumo de energía de los hogares. La introducción de la arquitectura lambda permite manejar conjuntos de datos masivos, actualizados cada hora, para construir modelos predictivos del consumo energético para cada casa.

(Martínez-Prieto et al., 2014) aplican la arquitectura lambda para manejar datos semánticos con el propósito de gestionar descripciones semánticas RDF a gran escala y establecen analogía entre las capas descritas por (Marz & Warren, 2015) y las existentes en la arquitectura SOLID. La ventaja que aporta la aplicación de esta arquitectura es que se obtienen resultados con baja latencia.

(Tian et al., 2014) construyen un sistema denominado (DiNoDB) aplicando la arquitectura lambda, que permite realizar consultas interactivas sobre sistemas de datos masivos (Big Data). En la evaluación del rendimiento se observa que DiNoDB responde a consultas interactivas con mejores resultados que otros sistemas similares (Hive, HadoopDB, Shark).

En el momento de escribir esta memoria no se han encontrado trabajos en el estado del arte que apliquen la arquitectura lambda al clustering.

2.3 Representación de los datos

La representación es una tarea fundamental para ciertos tipos de procesamiento automático y tiene como objetivo reducir la complejidad de los documentos y hacer que sean más fáciles de manejar. La representación debe ser fiel al contenido del documento y debe ir enfocada a que los algoritmos que se empleen posteriormente puedan extraer conocimiento útil de ella.

Dentro del conjunto de técnicas de representación de documentos, existen los modelos de representación vectoriales que han sido muy empleados en sistemas de clasificación y clustering de documentos. Las representaciones vectoriales resultan muy sencillas y descansan sobre la premisa de que el significado de un documento puede derivarse de un vector con el conjunto de rasgos presentes en el mismo (Fresno, 2006).

Las coordenadas de estos vectores se pueden construir a partir de múltiples funciones de ponderación, por ejemplo: frecuencia binaria (término presente o ausente), frecuencia del término (TF) que es el número de veces que aparece el término en el documento o frecuencia del término x frecuencia inversa del documento (TF-IDF) que veremos con detalle en la sección 2.3.1.

2.3.1 Modelo de espacio vectorial

El Modelo de Espacio Vectorial (VSM, Vector Space Model) (Salton et al., 1975) es el modelo básico de representación utilizado para clustering de documentos. Muchos modelos modificados se basan en él (Soucy & Mineau, 2005).
En este modelo cada documento se representa como un vector:

$$d_j = [a_{1j} \quad \dots \quad a_{Vj}]$$

donde cada elemento a_{ij} de la matriz documento-término queda definido como (Salton & Buckley, 1988):

$$a_{ij} = l_{ij} * g_i * n_j^{-1}$$

Donde:
l_{ij} es el peso local del término *i* en el documento *j*.
g_i es el peso global del termino *i* en la colección de documentos.
n_j^{-1} es el factor de normalización.

Es posible utilizar distintas funciones de pesado, pero usualmente se utiliza la frecuencia del término (TF) para ponderar el componente del peso local. Conjuntamente se suele utilizar la frecuencia inversa del documento (IDF) en el componente de peso global y no se aplica factor de normalización (valor 1).

La frecuencia inversa del documento (IDF) es el logaritmo entre el número de documentos del corpus y la frecuencia del documento (DF).

$$idf_i = idf(t_i) = log \frac{|D|}{|d \in D: t_i \in d|}$$

En resumen, es común representar cada documento mediante el siguiente vector:

$$d_j = \left[(tf_{1j} \cdot idf_1), (tf_{2j} \cdot idf_2), \dots, (tf_{Vj} \cdot idf_V)\right]$$

Aplicado a todos los documentos del corpus obtenemos finalmente una matriz TF-IDF:

$$tfidf_{ij} = tf_{ij} \cdot idf_i$$
$$i = 1,2, \dots V; j = 1,2, \dots, D$$

2.3.2 La maldición de la dimensionalidad

La maldición de la dimensionalidad es una expresión acuñada por Bellman en 1961 y se refiere al hecho de que muchos algoritmos que trabajan bien en pocas dimensiones se convierten en intratables cuando la entrada tiene un elevado número de dimensiones (Domingos, 2012).

La causa común de estos problemas es que cuando aumenta la dimensionalidad, el volumen del espacio aumenta exponencialmente haciendo que los datos disponibles se vuelvan dispersos. Esta dispersión es problemática para cualquier método que requiera significación estadística.

En el aprendizaje automático este efecto se advierte más. La generalización de forma correcta se convierte exponencialmente más difícil cuando la dimensionalidad de los datos crece, porque un conjunto de datos de entrenamiento de tamaño fijo cubre una fracción decreciente del espacio de entrada. Incluso con un tamaño moderado de 100 dimensiones y un conjunto de entrenamiento grande de un trillón de ejemplares, sólo se cubre una fracción de 10^{-18} del espacio de entrada (Domingos, 2012).

2.3.2.1 El valor de referencia y sus problemas

El clustering de documentos usando el algoritmo k-means con distancia coseno utilizando el Modelo del Espacio Vectorial (VSM) se ha considerado una referencia o línea base cuando se realiza comparación del rendimiento en cuanto a calidad del clustering. Este algoritmo es directo y fácil de implementar, pero también sufre de un elevado coste computacional que lo hace menos adecuado cuando trata con colecciones de documentos grandes. El problema de la maldición de la dimensionalidad se encuentra presente en el espacio de términos característico de un vocabulario. Por tanto se han desarrollado varias técnicas de reducción de la dimensionalidad para hacer mejoras sobre esa línea base (Xiao, 2010).

En la sección 2.4 se analizarán estas técnicas con más detalle.

2.4 Reducción de dimensionalidad

Aunque la complejidad y la velocidad de los algoritmos de clustering están relacionadas con el número de instancias en el conjunto de datos, por otra parte la propia dimensionalidad del conjunto de datos es otro aspecto importante. De hecho cuanto más dimensiones tienen los datos más complejidad y por tanto mayor tiempo de ejecución (Shirkhorshidi et al., 2014). Para reducir la complejidad y solventar el problema de la maldición de la dimensionalidad se utilizan técnicas de reducción de la dimensionalidad (Fodor, 2002).

Desde un punto de vista general, se pueden clasificar dos tipos de métodos de reducción (Parsons et al., 2004):

- selección de rasgos
- transformación de rasgos

En las técnicas de selección de rasgos, el objetivo no es tanto extraer nuevos rasgos sino eliminar aquellos que son irrelevantes para el modelo.

Las técnicas de transformación de rasgos intentan reducir la dimensionalidad a un número menor de dimensiones, que son combinación lineal o no lineal de las dimensiones originales. En estas técnicas el espacio de alta dimensionalidad se proyecta a un espacio con menor número de dimensiones.

Estos métodos han demostrado ser muy exitosos al descubrir estructuras latentes en los juegos de datos. Se han propuesto varias técnicas de transformación que incluyen:

- Análisis de Componentes Principales (PCA)
- Latent Semantic Analysis (LSA o LSI)
- Independent Component Analysis (ICA)
- Projection Pursuit
- Factor Analysis

LSA (Landauer et al., 1998) está relacionado con PCA con la diferencia de que PCA sólo se puede aplicar a matrices cuadradas mientras que LSA se puede aplicar a cualquier matriz (Milios, 2000).

En álgebra lineal es común referirse a LSA como Descomposición en Valores Singulares (Singular Value Descomposition, SVD).

La proyección SVD se calcula descomponiendo la matriz documento-término X_{txd} en forma de producto de tres matrices $T_{t\times n}$, $S_{n\times n}$ y $D_{d\times n}$.

$$X_{t\times d} = T_{t\times n}\, S_{n\times n} (D_{d\times n})^T$$

(Donde t es el número de términos, d es el número de documentos, $n=min(t,d)$ y T y D tienen columnas ortonormales, D^T denota la matriz traspuesta de D.

Las matrices T y D representan términos y documentos, respectivamente, en el nuevo espacio. La matriz diagonal S contiene los valores singulares en orden descendente.

El método ICA (Independent Component Analysis), a diferencia de PCA y LSA, es un método de orden superior que busca proyecciones lineales no necesariamente ortogonales entre sí, que son, estadísticamente, tan independientes entre sí como sea posible. El lector puede dirigirse a (Fodor, 2002) para más detalles.

La complejidad de SVD frente al número de documentos ($O(D^3)$) lo hacen inadecuado para conjuntos de datos grandes como los presentes en Big Data (Xiao, 2010).

(Milios et al., 2000) utilizan TF-IDF como esquema de ponderación de rasgos con una reducción de dimensionalidad seguido de un clustering mediante k-means. Se centran en las técnicas de reducción de la dimensionalidad y realizan una comparativa entre PCA, LSA y una técnica basada en la frecuencia del documento antes de realizar clustering con k-means y se observa que las dos primeras obtienen mejores resultados en la calidad frente a la tercera y que la representación con palabras rinde mejores resultados que representaciones mediante frases o N-Grams.

Las técnicas de transformación "Latent Dirichlet Allocation" (LDA) y "Probabilistic Latent Semantic Analysis" (PLSA) son modelos generativos[8] que se han utilizado ampliamente en tareas de reducción de dimensionalidad y para descubrir temáticas en textos (Chen & Liu, 2014). Ambas técnicas se analizan con detalle en 2.4.1 y 2.4.2. Los modelos obtenidos por ambas técnicas son equivalentes bajo una distribución de Dirichlet a priori uniforme (Girolami et al., 2003).

2.4.1 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) es un modelo generativo probabilístico que permite que conjuntos de observaciones puedan ser explicados por grupos no observados, los cuales explican por qué algunas partes de los datos son similares. La idea básica es que los documentos se representan como mezclas aleatorias sobre categorías o temáticas latentes, donde cada temática se caracteriza mediante una distribución sobre las palabras (Blei et al., 2003).

LDA asume el siguiente proceso generativo para cada documento d en un corpus D, tal que $D=\{d_1,d_2, \ldots, d_M\}$, y cada documento se representa como una secuencia de palabras $d_i=\{w_1,w_2,\ldots,w_n\}$.

Para cada documento d:

- Elegir N ~ Poisson (ξ)
- Elegir θ ~Dirichlet (α)
- Para cada una de las N palabras w_n :
  - Elegir una temática z_n=Multinomial(θ)
  - Elegir una palabra w_n con $p(w_n|z_n\beta)$ una probabilidad multinomial condicionada sobre la temática z_n

En la Figura 2.9 se representa gráficamente el modelo:

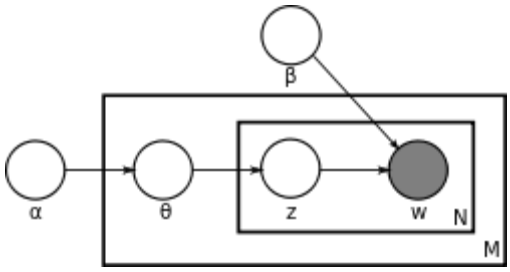

Figura 2.9. Representación gráfica del modelo de LDA. Fuente (Blei et al., 2003)

[8] El objetivo de muchas aplicaciones de aprendizaje automático es predecir el valor de un vector y (comúnmente la clase) dado el valor del vector de entrada con rasgos x. Desde una perspectiva probabilística el objetivo es encontrar la distribución condicional $p(y|x)$. La aproximación más común a este problema es representar la distribución condicional usando un modelo paramétrico y determinar los parámetros usando un juego de datos de entrenamiento de pares $\{x_i, y_i\}$. La distribución de probabilidad resultante se puede usar para hacer predicciones de y para nuevos valores de x. Esta es una aproximación **discriminativa**, porque la distribución condicional discrimina directamente entre los diferentes valores de y.
Una aproximación alternativa, consiste en encontrar la distribución conjunta $p(y,x)$ expresada como un modelo paramétrico y entonces posteriormente usar esta distribución conjunta para evaluar la probabilidad condicional $p(y|x)$ para realizar predicciones de y para nuevos valores de x. Esta aproximación, que es la que se utiliza en LDA y PLSA, se conoce como modelo **generativo**. (Lasserre et al., 2006).

En la figura las cajas representan réplicas. La caja exterior representa los distintos documentos, mientras que la interior representa la repetición en la elección de temáticas y palabras dentro de un documento.

Donde:

α: es el parámetro de Dirichlet a priori en la distribución de temáticas por documento.

β: es el parámetro de Dirichlet a priori en la distribución de temáticas por palabra.

θ_i: es la distribución de temáticas para el documento *i*.

$z_{i,j}$: es la temática para la palabra *j* en el documento *i*.

$w_{i,j}$: es la palabra *j* en el documento *i*.

$w_{i,j}$ son variables observables, mientras que el resto de variables son latentes, son las temáticas ocultas.

La probabilidad total del modelo es:

$$P(w|\theta,\beta) = \sum_{z} P(w|z,\beta) \cdot P(z|\theta)$$

El algoritmo que se utiliza para estimar LDA de forma empírica es Expectativa-Maximización (Expectation-Maximization, EM) que se basa en maximizar la función log-verosimilitud (log likelihood) con respecto a $\alpha\ y\ \beta$.

$$\ell(\alpha,\beta) = \sum_{d=1}^{M} \log p(w_d|\alpha,\beta)$$

La función de probabilidad no se puede calcular directamente, pero podemos maximizar la función de verosimilitud calculando aproximaciones a través de parámetros variacionales: $\{\gamma_d^*, \phi_d^*: d\ \epsilon D\}$ de esta forma:

$$\phi_{ni} \propto \beta_{iw_n} \exp\{E_q(\log(\theta_i)\,|\gamma)\}$$

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni}$$

Donde:

$$E_q(\log(\theta)\,|\gamma) = \Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)$$

Donde Ψ es la primera derivada de la función $\log(\Gamma)$ que se puede calcular a través de aproximaciones de Taylor.

Para ello se definen dos pasos E y M para ejecutar iterativamente (Asuncion et al., 2009).

- Paso-E: Para cada documento encuentra los valores óptimos sobre los parámetros variacionales $\{\gamma_d^*, \phi_d^*: d\ \epsilon D\}$.
- Paso-M: Maximiza sobre la función de verosimilitud con respecto a los parámetros $\alpha\ y\ \beta$.

Se ha estudiado y comparado en (Banerjee & Basu, 2007) la reducción de dimensionalidad en conjuntos de datos textuales utilizando técnicas como LDA, Dirichlet Compound Multinomial (DCM) y Mixture of von Mises-Fisher (vMF). Las técnicas utilizadas en el citado trabajo se muestran adecuadas para la reducción de dimensionalidad en esos contextos y para encontrar temáticas latentes. Además, estas técnicas encuentran utilidad en varias aplicaciones de la minería de textos como la organización de documentos

en gestión de contenidos, recuperación de información y filtrado en servicios agregadores de noticias en internet. Los trabajos que estudian las técnicas (Banerjee & Basu, 2007)(Xiao, 2010) concluyen que LDA es un buen modelo para encontrar temáticas a nivel de palabra mientras que vMF encuentra de forma más eficiente las relaciones de la temática con el documento.

(Xiao, 2010) realiza una propuesta de reducción de dimensionalidad seguido de clustering con k-means. La aplicación conjunta de estas técnicas para obtener finalmente clustering sobre contextos Big Data (datos de redes sociales) implica un desafío en lo relacionado con la velocidad de proceso, que propone resolver mediante Map-Reduce para paralelizar el procesamiento.

En (Bíró et al., 2009) utilizan LDA aplicado a técnicas de filtrado de spam, y realizan un estudio con varias formas de representación como entrada de LDA, y concluyen que aquellas que obtienen mejor calidad en la clasificación son las que utilizan TF-IDF como forma de representación previa a LDA.

2.4.2 Probabilistic Latent Semantic Analysis (PLSA)

Probalistic Latent Semantic Analysis (PLSA) es otro modelo de reducción de dimensionalidad ampliamente usado sobre documentos, introducido por (Hofmann, 1999). A continuación (Figura 2.10) se muestra la representación gráfica del modelo PLSA.

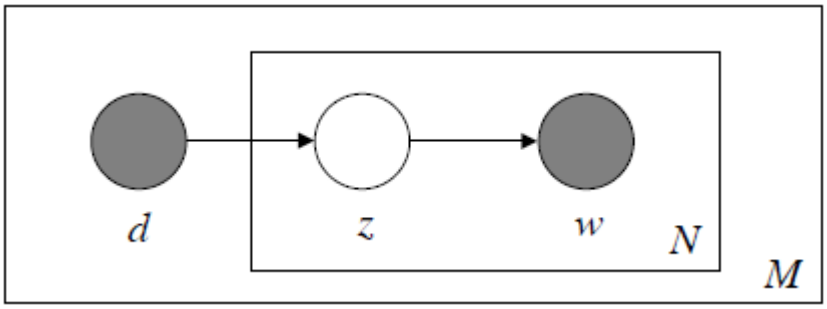

Figura 2.10. Representación gráfica del modelo PLSA. Fuente (Blei et al, 2003)

El modelo supone que un documento d y una palabra w_n son condicionalmente independientes dada una temática oculta z. La probabilidad del modelo es:

$$P(d, w_n) = \sum_z P(z) \cdot P(w_n|z) \cdot P(d|z) = P(d) \sum_z P(w_n|z) \cdot P(z|d)$$

En este caso se intenta maximizar la función log-verosimilitud definida de esta forma:

$$\ell = \sum_{d=1}^{M} \sum_{w=1}^{W} n(d, w) \log P(d, w)$$

El procedimiento de estimación se realiza mediante el algoritmo de Expectation-Maximización (EM) que alterna dos pasos.

En el paso E se calcula:

$$P(z|d, w) = \frac{P(z) \cdot P(d|z) \cdot P(w|z)}{\sum_{z'} P(z') \cdot P(d|z') \cdot P(w|z')}$$

$P(z|d,w)$ es la probabilidad de que una palabra en un documento en particular d sea explicada por el factor correspondiente a z.
En el paso M se calcula:

$$P(w|z) = \frac{\sum_d n(d,w) \cdot P(z|d,w)}{\sum_{d,w'} n(d,w') \cdot P(z|d,w')}$$

$$P(d|z) = \frac{\sum_w n(d,w) \cdot P(z|d,w)}{\sum_{d',w} n(d',w) \cdot P(z|d',w)}$$

$$P(z) = \frac{1}{R} \sum_{d,w} n(d,w) \cdot P(z|d,w)$$

Donde:

$$R = \sum_{d,w} n(d,w)$$

El algoritmo alternativamente ejecuta los pasos E y M hasta que se alcanza convergencia y encuentra un máximo local de la función log-verosimilitud.

(Hoffman 2001) realiza un estudio utilizando PLSA sobre colecciones textuales para reducir la elevada dimensionalidad presente en los documentos. El espacio con dimensión reducida obtenido, presenta mejor calidad de clustering comparado frente a otras técnicas similares (LSA).

2.4.3 PLSA y LDA en Big Data

En (Wang et al., 2009) nos presentan una versión paralela de LDA denominada PLDA, y en su trabajo realizan dos implementaciones para comparar cómo escalan: una basada en Message Passing Interface MPI (concretamente en mpich) y otra en Map-Reduce. Obtiene mejores resultados en MPI porque usa una comunicación eficiente llevada a cabo en memoria frente a la implementación en Hadoop que implica operaciones de entrada/salida en disco entre iteraciones.

Por otra parte, se han realizado varios trabajos para estudiar PLSA en contextos Big Data frente a la escalabilidad:

En (Jin et al., 2011) los autores realizan dos implementaciones (P2PLSA y P2PLSA+) utilizando el clásico algoritmo EM que implica un proceso iterativo con dos pasos alternativos hasta que se obtiene convergencia. La implementación se ha realizado en Map-Reduce de Hadoop y la conclusión del trabajo es que es posible trabajar bien con conjuntos de datos grandes (MovieLens) sin embargo los autores admiten que los algoritmos presentan ciertas limitaciones y problemas de escalabilidad cuando se supera cierto nivel en el tamaño de los datos de entrada.

En (Li et al., 2012), realizan una implementación paralela de PLSA basado en Map-Reduce que denominan PPLSA y usan scaleup, sizeup y speedup para evaluar el rendimiento. La implementación presenta el inconveniente de la limitación de memoria, por eso el algoritmo no trabaja bien cuando el juego de datos es demasiado grande.

En (Zhao Y. et al., 2014) intentan sortear los problemas citados en el trabajo anterior mediante una reorganización de los factores, de forma que llegue información de un único documento a cada nodo. Sin embargo todavía hay datos intermedios que se tienen que replicar en cada nodo.

2.5 Clustering en Big Data

El clustering es la clasificación no supervisada de patrones (observaciones, elementos de datos o rasgos de un vector) en grupos que conocemos como clusters. Por tanto trata de agrupar entidades en base a cierta noción de similitud (Jain 1999).
Como ya hemos visto, el tamaño de los datos crece de forma explosiva y la complejidad intrínseca de los algoritmos de clustering es elevada. La escalabilidad es un desafío permanente en el clustering (Wu et al., 2014).

Hay muchos algoritmos de clustering en Big Data. En esta sección se presenta una categorización que incluye varias categorías según se encuentran en la literatura (Berkhin, 2002) (Xu & Wunsch, 2005) (Fahad et al., 2014). La categorización propuesta sigue el punto de vista de un diseñador de algoritmos que se enfoca en los detalles técnicos del procedimiento de clustering.
Seguidamente se enumeran las distintas categorías:

- **Basados en particionamiento**: Dividen los objetos de datos en una serie de particiones, donde cada partición representa un cluster. Estos clusters deben cumplir una serie de requisitos: cada grupo debe contener al menos un objeto y cada objeto debe pertenecer solamente a un grupo. En k-means, por ejemplo un centro es el promedio de todos los puntos y las coordenadas representan la media aritmética. Ejemplos de este tipo: k-means, k-medoids, k-modes, PAM, CLARA, CLARANS y FCM.
- **Basados en jerarquías**: Los datos se organizan de forma jerárquica en función de su proximidad. La proximidad se obtiene a partir de los nodos intermedios. Un dendrograma representa el conjunto de datos completo, mientras que los datos individuales se representan mediante nodos hoja. El cluster inicial gradualmente se divide en varios clusters según se avanza en la jerarquía. Los métodos de clustering jerárquico pueden ser aglomerativos (de abajo a arriba) o divisivos (de arriba abajo). Un algoritmo aglomerativo empieza con un objeto en cada cluster y recursivamente va aglutinando entre sí aquellos clusters que son más apropiados. Un algoritmo divisivo empieza con un solo cluster sobre todo el conjunto de datos y recursivamente va dividiendo los clusters más adecuados. El proceso continúa hasta que se alcanza un criterio de parada (frecuentemente cuando se alcanza el número esperado de clusters). El método jerárquico tiene una desventaja importante: una vez que se ha realizado un paso (mezclar o dividir) ya no se puede deshacer.
  Ejemplos de algoritmos conocidos de esta categoría son BIRCH, CURE, ROCK y Chameleon.
- **Basados en densidad:** Aquí los objetos presentes en los datos se separan basándonos en sus regiones de densidad, conexión y frontera. Un cluster se define como un componente densamente conectado que crece en cualquier dirección

donde le lleve la densidad. Por tanto, los algoritmos basados en densidad son capaces de descubrir formas arbitrarias. También esto ofrece una protección natural contra valores atípicos. DBSCAN, OPTICS, DBCLASD y DENCLUE son algoritmos de esta categoría que usan métodos para filtrar ruido (datos atípicos) y descubrir clusters de formas arbitrarias.

- **Basados en rejilla:** En este caso el espacio de los datos se divide en rejillas. La principal ventaja de esta aproximación es su rápido tiempo de proceso, porque realiza una pasada a través del conjunto de datos para calcular los valores estadísticos en las rejillas. El rendimiento de los métodos basados en rejilla depende del tamaño de la rejilla, que normalmente es menor que el tamaño del conjunto de datos. Sin embargo, para distribuciones de datos muy irregulares, el uso de un tamaño de rejilla uniforme puede no ser suficiente para obtener la calidad de clustering necesaria. Wave-Cluster y STING son ejemplos típicos de algoritmos de clustering de esta categoría.
- **Basados en modelo:** Este método optimiza el ajuste entre los datos y un modelo matemático definido. Se basa en suponer que los datos se generan a través de una mezcla de distribuciones de probabilidad subyacentes. También, lleva a una forma de determinar automáticamente el número de clusters basándose en estadística estándar, teniendo en cuenta el ruido y por tanto rindiendo un método de clustering robusto. Principalmente hay dos aproximaciones que se basan en métodos basados en modelo: aproximaciones estadísticas y de redes neuronales. Ejemplos de algoritmos basados en modelo podemos citar Expectativa-Maximización (EM, Expectation-Maximization), MCLUST, COBWEB y aproximaciones con redes neuronales (con mapas autoorganizativos). Las aproximaciones estadísticas usan medidas de probabilidad para determinar conceptos o clusters.

Por otra parte, y desde el punto de vista de Big Data, para enfrentarse al problema de crecimiento de datos, teniendo en cuenta la escalabilidad, hay tres estrategias básicas en clustering: (Aggarwal, 2013)

1. Reducir el número de iteraciones mediante algoritmos de una pasada.
2. Técnicas para reducir la complejidad del tamaño de los datos de entrada (Ejemplo: Singular Value Descompositon).
3. Algoritmos paralelos y distribuidos (Ejemplo k-means con Map-Reduce).

Una tendencia indicada por la bibliografía es combinar varias de estas técnicas para alcanzar mejor escalabilidad (Aggarwal, 2013). En el presente trabajo se combinan dos de ellas, (1) disminución la complejidad de los datos de entrada mediante reducción de dimensionalidad y (2) ejecución distribuida. En la siguiente sección (2.5.1) se estudia el algoritmo k-means||, que es el que se utiliza experimentalmente en este trabajo, y que afronta el problema del crecimiento de datos mediante la tercera estrategia.

2.5.1 k-means en Big Data

k-means es un algoritmo iterativo que redistribuye cada punto a su centroide más cercano (Friedman et al., 2001).

El algoritmo k-means recibe un conjunto de datos consistente en unos puntos $(x_1, x_2, .., x_n) \epsilon \mathbb{R}^d$ y trata de particionar estas observaciones en k centros $\mathcal{C}$ que minimicen la suma de cuadrados dentro de cada grupo:

$$argmin_s \sum_{i=1}^{k} \sum_{x_j \epsilon C_i} \|x_j - \mu_i\|^2$$

μ_i: es la media de puntos en el cluster.
El algoritmo consiste en (Figura 2.11):

1) *Asignar cada punto a un cluster basándose en cuál es su centro más cercano.*
2) *Reemplazar cada centro con el promedio de los puntos en el cluster.*

Figura 2.11. Algoritmo de k-means

Existen comparativas en la bibliografía que estudian la calidad del cluster utilizando distintas técnicas de clustering que incluyen k-means. Se observa que esta técnica es más rápida comparada con algoritmos jerárquicos (Steinbach et al., 2000).
(Arthur & Vassilvitskii, 2007) realizan una modificación del algoritmo k-means que denominan k-means++ que se basa en elegir los centros de una forma específica y obtienen un rendimiento O(log k)-competitivo respecto al clustering óptimo (k es el número de clusters). Intuitivamente, la forma de elegir los centros es intentar que sean dispersos: el primer centro se elige al azar y los siguientes centros se eligen al azar de entre los puntos restantes con probabilidad proporcional al cuadrado de la distancia desde cada punto hasta el centroide más cercano.

(Bahmani et al., 2012) nos indican que una desventaja de k-means++ es su naturaleza inherentemente secuencial, que limita su aplicación sobre datos masivos: se deben hacer k pasadas sobre los datos para encontrar un buen conjunto de centros inicial y propone como alternativa el algoritmo k-means|| que reduce drásticamente el número de pasadas necesarias debido a que se realiza una buena inicialización. Este es el algoritmo que se encuentra implementado en Apache Spark que se ha utilizado en este trabajo en la capa batch. El pseudocódigo se muestra en la Figura 2.12.

El coste del algoritmo k-means|| es comparable al de k-means++, pero el primero requiere ejecutarse en menos ciclos, lo cual lo convierte en más rápido. Por tanto el coste computacional de ambos es O(log-k)-competitivo (Bahmani et al., 2012) y el tiempo de ejecución mejora linealmente con el número de máquinas disponibles (Dhillon & Modha, 2000).

Los autores (Bahmani et al., 2012) realizan una implementación del algoritmo, para su ejecución de forma paralela en Map-Reduce, subdividiendo los pasos en fases map y reduce. Obtienen, en cuanto a rendimiento, resultados mejores que k-means++ debido a que son necesarias menos iteraciones. Se pueden encontrar más detalles en (Bahmani et al., 2012).

Entradas:

a) *X: conjunto de puntos*
b) *k: número de clusters*
c) *l: factor de sobremuestreo*

Pseudocódigo:

1. $C \leftarrow$ *centro muestreado a partir de un punto al azar de X*
2. $\psi = \phi X(C)$ *//coste del clustering de X respecto a C*
3. ***for*** $O(\log(\psi))$ ***veces do***
   $C' \leftarrow$ *muestrea cada punto* $x \in X$ *independientemente con probabilidad* $p_x = \frac{l\, d^2(x,C)}{\phi X(C)}$
4. $C \leftarrow C \cup C'$
5. ***end for***
6. *asignar pesos a los puntos en C*
7. *recluster de los puntos ponderados en C en k clusters*

Figura 2.12. Pseudocódigo del algoritmo k-means| |

En (Zhao et al., 2009) se realiza un estudio de la escalabilidad de una implementación paralela de k-means en Hadoop denominada PKMeans. La implementación del algoritmo k-means se realiza mediante Map-Reduce y trabaja sobre conjuntos de datos de hasta 8Gb. El estudio analiza la escalabilidad a través de la evaluación de scaleup, speedup y sizeup y concluye que el algoritmo propuesto puede procesar conjuntos de datos en el contexto de Big Data de forma efectiva sin necesidad de hardware sofisticado.

2.6 Flujo de datos (Streaming)

Hoy en día existen muchas aplicaciones que producen grandes volúmenes de información en intervalos de tiempo muy corto (flujos de datos). Algunos ejemplos son: datos de sensores, redes sociales, blogs, flujo de transacciones de tarjetas de crédito, log de eventos en aplicaciones o redes, mercado de valores, etc. Esto causa limitaciones en la capacidad de almacenamiento y tiempo de procesamiento. A pesar de estas características, la necesidad de procesar los datos sigue vigente y se requieren algoritmos que actúen en entornos dinámicos donde los datos se recojan a través del tiempo.

Una definición formal de flujo de datos (o streaming) es aquella secuencia de objetos de datos $x_1, x_2, \ldots x_n$, que representamos como $S = \{x_i\}_{i=1}^{N}$, donde $(N \rightarrow \infty)$.

Los algoritmos sobre flujos de datos tienen que diseñarse para que puedan tratar con unas propiedades inherentes de los flujos (Henzinger et al., 1998):

- Los objetos de datos llegan continuamente.
- El tamaño del flujo puede ser sin límite y potencialmente infinito.
- Sólo se puede acceder a los datos en el orden de llegada.
- El acceso aleatorio a los datos no está permitido.
- Se supone que la memoria es pequeña respecto al número de puntos.
- La fuente de datos puede evolucionar.

Los datos de los flujos no se suelen almacenar, por consiguiente sólo se procesan y almacenan resúmenes de estos datos.

Se han desarrollado técnicas para tratar con estas limitaciones como las ventanas deslizantes que es un método que consiste en una ventana que mantiene los últimos *N* elementos que han llegado y ejecuta el algoritmo sobre esos elementos. Se mantiene el resumen y se descartan los elementos, así siempre hay estadísticas actualizadas (Rajaraman & Ullman, 2012).

Los algoritmos sobre flujos de datos se han estudiado extensamente por los investigadores y hay una amplia literatura sobre las diferentes técnicas (Gaber et al., 2005) y (Gama et al., 2007).

El funcionamiento de varias tecnologías que trabajan sobre flujos (Apache Storm, Apache Spark) está basado en el procesamiento de micro-lotes (Marz & Warren, 2015).

En el caso concreto de Spark (Figura 2.13), el sistema recibe un flujo de datos de entrada y lo divide en lotes, que se procesan para generar el flujo final de resultados en lotes procesados.

Figura 2.13. Esquema del flujo de datos en Spark (Fuente http://spark.apache.org)

2.6.1 Clustering incremental

Una de las operaciones que se puede realizar sobre flujos de datos es el clustering, que permite obtener información de agrupación de los datos en tiempo real.

(O'Callaghan et al., 2002) presentan un algoritmo de clustering en streaming de una pasada, basado en k-medoids, que da una aproximación en factor constante, es decir, encuentra una solución no-óptima pero aproximada en complejidad computacional constante (Θ(1)). El principal problema de este algoritmo desde el punto de vista práctico es que el tiempo de procesamiento promedio por elemento es muy grande. Es proporcional a la cantidad de memoria utilizada que es polinomial-logarítimico en *n*. Esto lo hace indeseable en escenarios donde necesitamos procesar datos rápidamente como es el caso que nos ocupa.

Para evitar ese inconveniente, (Ailon et. al. 2009) presentan un algoritmo alternativo, basado en k-means, que está disponible en Spark y que es el que se utiliza en este trabajo como uno de los pasos de la capa speed para procesar el flujo de nuevos documentos.

En la Figura 2.14 se muestra el pseudocódigo del algoritmo de streaming de (Ailon et. al. 2009). El rendimiento de este algoritmo es $O(c^{\alpha}\log(k))$, donde $\alpha \approx log\ n/\log M$. *n* es el número de puntos de entrada y *M* es la cantidad de memoria de trabajo disponible para el algoritmo. Este algoritmo utiliza de forma combinada otros dos algoritmos, k-means++ y k-means#, en una estrategia divide y vencerás propuesta en (Guha et al., 2003).

En la Figura 2.15 se muestra el pseudocódigo de k-means#. El algoritmo k-means# consiste en una modificación realizada sobre el algoritmo kmeans++ que obtiene O(log(k)) centros en tiempo O(1)-competitivo. La modificación consiste en que cada ciclo en lugar de elegir un único centro se eligen O(log k) centros (Ailon et al., 2009).

Entradas:

a) *Conjunto de puntos* $S \subset R^d$ *. Sea* $n = |S|$
b) *Número de clusters deseado* $k \in N$
c) *A, un algoritmo de aproximación (a,b) al objetivo de k means*
d) *A' un algoritmo de aproximación (a',b') al objetivo de k means*

Pseudocódigo:

1. *Divide* S *en grupos* $S_1, S_2, S_3, \dots, S_l$
2. ***foreach*** $i \in \{1,2,3, \dots, l\}$ ***do***
3. *Ejecuta A sobre* S_i *para obtener* $\leq ak$ *centros* $T_i = \{t_{i1}, t_{i2}, \dots\}$
4. *Designa los clusters inducidos de* S_i *como* $S_{i1} \cup S_{i2} \cup \dots$
5. ***end foreach***
6. $S_w \leftarrow T_1 \cup T_2 \cup \dots \cup T_l$, con pesos $w(t_{ij}) \leftarrow |S_{ij}|$
7. *Ejecuta A' sobre* S_w *para obtener* $\leq a'k$ *centros* T
8. ***return*** T

(A y A' se utilizan algoritmos k-means ++ y k-means # para llevar a cabo una estrategia divide y vencerás)

Figura 2.14. Pseudocódigo del algoritmo Streaming k-means

Pseudocódigo:

1. *Elegir 3·log(k) centros independientes y uniformes al azar a partir de X*
2. ***repetir*** *(k-1) veces*
3. *Elegir (3·log(k)) centros independientes con probabilidad* $\frac{D(x')^2}{\sum_{x \epsilon X} D(x)^2}$
4. ***end*** *repetir*

Figura 2.15. Pseudocódigo del algoritmo k-means#

En el clustering incremental se mantiene un resumen de los datos y se pueden descartar los elementos procesados del flujo. Es decir, para cada lote de datos en el flujo, se determinan los nuevos centroides y se actualizan de la siguiente forma:

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

$$n_{t+1} = n_t + m_t$$

Donde:

c_t: es el centro anterior del cluster.

n_t: es el número de puntos asignados al cluster hasta ahora.

x_t: es el nuevo centro determinado para el lote actual.

m_t: es el número de puntos añadidos al cluster en el lote actual.

α: es el factor de decaimiento, se puede usar para ignorar los datos pasados (usando α=1 se usarán todos los datos desde el principio).

2.7 Métodos de evaluación

2.7.1 Evaluación de la escalabilidad

Se han utilizado diferentes técnicas a la hora de evaluar la medida del rendimiento en procesamiento paralelo que se encuentran en la bibliografía (Englert et al., 1989)(Smith et al, 1989).
Se detallan a continuación:

- Scaleup
- Speedup
- Sizeup

Estas medidas pueden expresarse en función de la latencia, que es el tiempo para terminar una tarea, y del throughput o número de tareas ejecutadas en una cantidad de tiempo.

La magnitud **scaleup** (escalabilidad) se define como la habilidad de un sistema *n* veces mayor que otro de llevar a cabo una tarea *n* veces mayor en el mismo tiempo de ejecución que el sistema original. Se define con la siguiente fórmula:

$$scaleup = \frac{T_1}{T_{n,n}}$$

Donde T_1 es el tiempo de ejecución para un core, y $T_{n,n}$ es el tiempo de ejecución para procesar un conjunto de datos *n* veces mayor sobre *n* cores. Cuando existe buena escalabilidad podemos obtener tiempo de procesamiento constante añadiendo nodos, aunque crezca igualmente el conjunto de datos.

La magnitud **speedup** mide cuantas veces es más rápido un sistema paralelo frente a su algoritmo secuencial correspondiente (ejecutándose en un solo core). Se define mediante la siguiente fórmula:

$$speedup = \frac{T_1}{T_P}$$

Donde T_1 es el tiempo de ejecución en un core, mientras que T_P es el tiempo de ejecución para procesar el mismo conjunto de datos sobre *p* cores. Un valor mayor de la medida representa una mejor respuesta del sistema usando paralelismo.
A partir de esta se puede calcular la eficiencia en paralelización como:

$$eficiencia = \frac{speedup}{p}$$

Un sistema que escala linealmente tendrá una eficiencia de 1.

Sizeup mide cuánto tiempo lleva ejecutarse en un mismo sistema un conjunto de datos *m* veces mayor comparado frente al tiempo de ejecución del conjunto de datos original. Se define mediante la siguiente fórmula:

$$sizeup = \frac{T_m}{T_1}$$

Donde T_1 es el tiempo de ejecución para procesar un conjunto de datos, mientras que T_m es el tiempo de ejecución para procesar un tamaño de datos *m* veces mayor.

2.7.2 Evaluación de la calidad

Para evaluar el rendimiento del clustering desde el punto de vista de la calidad se han utilizado en este trabajo métricas que son habituales. Son las siguientes: Medida-F (Larsen & Aone, 1999), Pureza (Hotho et al., 2003) e Información Mutua Normalizada (Zhong, 2005).

Medida-F

Se puede expresar en función de los verdaderos positivos, falsos positivos y falsos negativos según la Tabla 2.2.

$$medidaF = \frac{2 \cdot true\_positive}{2 \cdot true\_positive + false\_negative + false\_positive}$$

O en función de la precisión y la cobertura (recall):

$$medidaF = \frac{2 \cdot precisión \cdot recall}{precisión + recall}$$

Siendo:

$$precisión = \frac{relevantes \cap recuperados}{recuperados}$$

$$recall = \frac{relevantes \cap recuperados}{relevantes}$$

Una aproximación común para evaluar clustering es estudiar las relaciones entre cada par de puntos en el conjunto de datos, obteniendo un conjunto de $n*(n-1)/2$ pares para n objetos (Achtert et al., 2012). Así podemos determinar los verdaderos positivos, falsos positivos y falsos negativos obtenidos como se muestra en la Tabla 2.2.

Pares de puntos	Mismo cluster	Distinto cluster
Misma clase	true_positive	false_negative
Distinta clase	false_positive	true_negative

Tabla 2.2. Asignación de pares de puntos en función de su pertenencia a una misma clase y/o cluster

Pureza

Para calcular la pureza de un cluster, también conocida como precisión del cluster (cluster accuracy) se calcula contando el número de objetos correctamente asignados a la clase mayoritaria de un cluster y dividiendo por el total. Formalmente:

$$Purity(\Omega, \mathcal{C}) = CA(\Omega, \mathcal{C}) = \frac{1}{N} \sum_{k} max_j \left| \omega_k \cap c_j \right|$$

Donde $\Omega = \{\omega_1, \omega_2, \dots, \omega_k\}$ es el conjunto de clusters y $\mathcal{C} = \{c_1, c_2, \dots, c_j\}$ es el conjunto de clases.

Información mutua normalizada, NMI (Normalized Mutual Information)
Es una métrica común de validación del clustering.

$$NMI(\Omega, \mathcal{C}) = \frac{MI(\Omega, \mathcal{C})}{[H(\Omega) + H(\mathcal{C})]/2}$$

Donde *MI* es información mutua:

$$MI(\Omega, \mathcal{C}) = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \cdot log \frac{N \cdot |\omega_k \cap c_j|}{|\omega_k||c_j|}$$

$H(\Omega)$ y $H(\mathcal{C})$ son respectivamente la entropía sobre los clusters y sobre las clases:

$$H(\Omega) = -\sum_k \frac{|\omega_k|}{N} \cdot log \frac{|\omega_k|}{N}$$

$$H(\mathcal{C}) = -\sum_j \frac{|c_j|}{N} \cdot log \frac{|c_j|}{N}$$

3 Diseño del método propuesto

En este capítulo se describe el método diseñado para la aplicación de la arquitectura lambda al clustering de documentos y se detallan los algoritmos utilizados.

3.1 Descripción

En esta sección se analizará la aplicación de la arquitectura lambda al problema de clustering de documentos sobre conjuntos de datos masivos y en evolución.

No se ha encontrado en la bibliografía referencias a esta aplicación en particular de la arquitectura lambda, por lo que esto supone la novedad del proyecto desde el punto de vista científico.

El problema que se intenta resolver con esta arquitectura es realizar el clustering de documentos en contextos Big Data, en los cuales hay una incorporación continua de nuevos documentos, y obtener actualización de los resultados en tiempo real. Los requisitos del diseño del software vienen marcados por las directrices de la arquitectura lambda, es decir, el diseño de las tres capas (batch, speed y serving), con el fin de evitar el problema de la latencia.

Básicamente se aplican los siguientes pasos consecutivos sobre los documentos:

- Representación de los documentos según el modelo de espacio vectorial.
- Reducción de la dimensionalidad.
- Clustering.

Esos tres pasos son los que conforman la capa batch y la capa speed. La capa batch realiza un procesamiento por lotes sobre toda la colección de documentos mientras que la capa speed únicamente sobre los documentos nuevos. La capa serving será responsable de ofrecer la información que han generado las otras dos capas. La Figura 3.1 esquematiza el diseño propuesto.

Los nuevos documentos incorporados en el sistema, por una parte sirven de entrada de la capa speed donde se procesan inmediatamente, y por otra se encolan en el conjunto de datos maestro hasta el próximo ciclo de procesamiento de la capa batch. Es en esta capa donde se almacenan, ya que en la capa speed después de procesarse se descartan.

La arquitectura lambda propone que se debe mantener un conjunto de datos maestro con la información sin ningún procesamiento ("cruda") de forma que permita regenerar por completo toda la información en caso de que se incorporen modificaciones o exista un fallo humano. Por tanto, el conjunto de datos maestro en este trabajo se ha considerado el corpus de documentos completo, sin ningún filtrado ni cálculo previo sobre cada documento. Este requisito tiene como consecuencia que se deban realizar todas las computaciones necesarias para obtener resultados. Es decir, calcular la representación del documento, reducir su dimensionalidad y ejecutar la fase de clustering. Como resultado obtenemos la relación de cada documento con su cluster y la información que caracteriza cada cluster como es su centroide.

Disponer de toda la información sin descartar nada permitiría en un futuro realizar modificaciones en los cálculos o en la representación (por ejemplo utilizar stemming, o n-Grams). La clave es no renunciar a los datos que en un futuro puedan ser necesarios.

Existen múltiples combinaciones de tecnologías para realizar la implementación. Por ejemplo se puede utilizar Apache Hadoop para la capa batch y Apache Storm para la capa speed, pero se ha tomado la decisión de utilizar Apache Spark para ambas capas para facilitar la integración de los datos entre capas y aprovechar el mismo API de programación en la medida de lo posible.

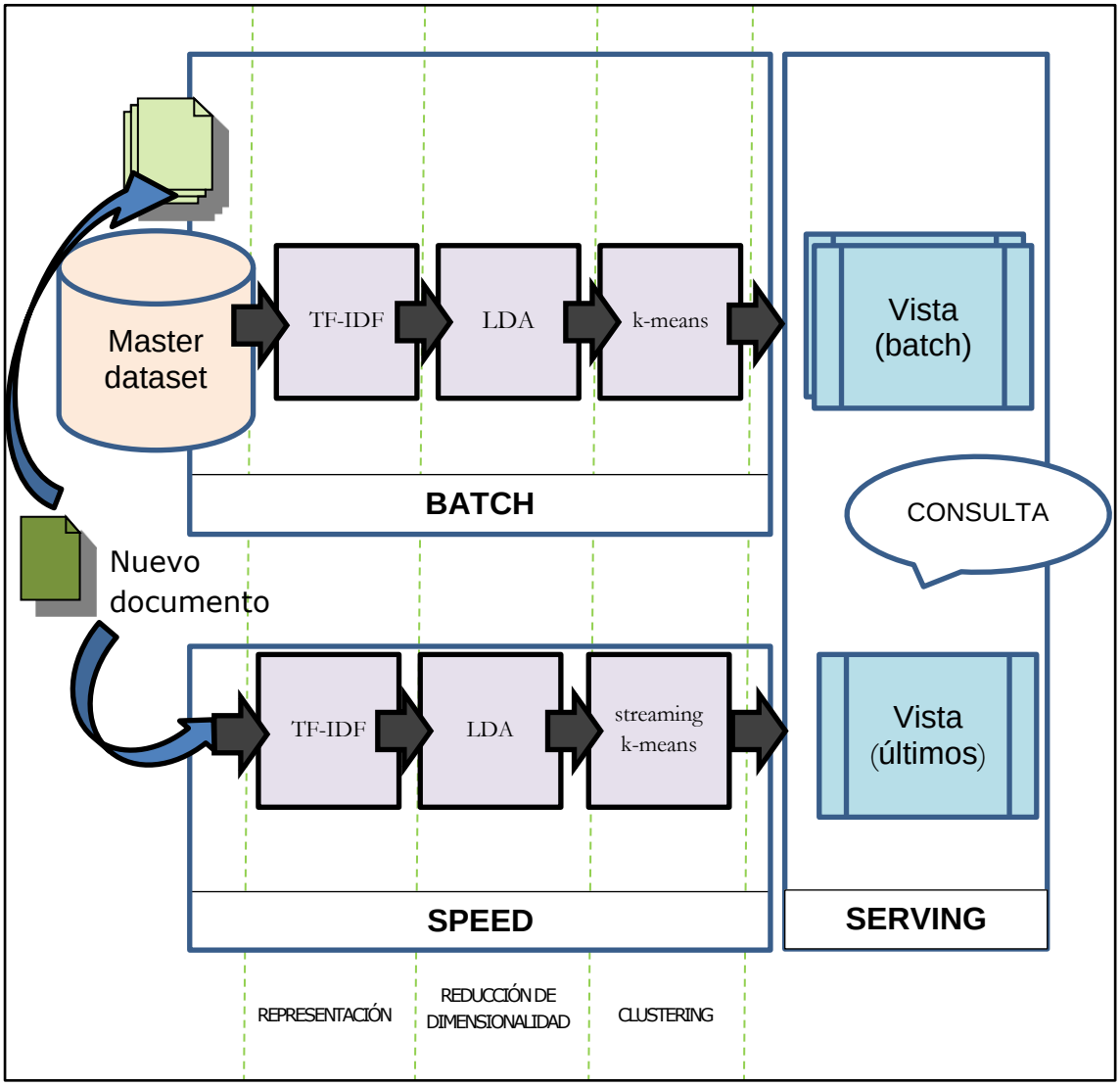

Figura 3.1. Arquitectura lambda aplicada a clustering de documentos

Como prueba de concepto para la capa proveedora (Serving Layer) se ha utilizado HIVE[9] por la facilidad para crear tablas a partir de la salida en ficheros de texto procedente de las otras dos capas.

[9] https://hive.apache.org/

Ante el problema de la elevada dimensioalidad, que se ha justificado en la sección 2.3.2, se ha aplicado una técnica de reducción de la dimensionalidad, concretamente Latent Dirichlet Allocation (LDA).

Para la entrada de LDA se ha utilizado la representación según el Modelo de Espacio Vectorial tal y como demuestran los buenos resultados obtenidos en (Bíró et al., 2009).

Respecto a la tarea de clustering de documentos se ha elegido el algoritmo ampliamente utilizado que es k-means concretamente la versión k-means||.

Para la capa speed se ha elegido un algoritmo de clustering incremental que permite actualizar las variaciones que puedan existir en los clusters con la llegada de nuevos documentos y se ha utilizado el algoritmo incremental de (Ailon et. al. 2009) descrito en la sección 2.6.1.

En la sección 3.4 se describirán en detalle el diseño de las tres capas de la arquitectura lambda presentes en esta propuesta.

3.2 Algoritmos utilizados

Esta sección recoge a modo de resumen los algoritmos utilizados en este trabajo. En la capa batch se han utilizado los algoritmos indicados en la Tabla 3.1 y en la capa speed los recogidos por la Tabla 3.2. Se han elegido éstos porque se encuentran presentes en el estado del arte y ofrecen buenos resultados.
Los detalles de los algoritmos se han estudiado en el capítulo 2.

	Algoritmo	Referencias	Sección
Clustering	k-means||	Bahmani B. et al., 2012	2.5.1
Reducción dimensionalidad	Latent Dirichlet Allocation	Blei et al, 2003 Asuncion et al., 2009	2.4.1

Tabla 3.1. Resumen de algoritmos utilizados en Batch Layer

	Algoritmo	Referencias	Sección
Clustering incremental	Streaming k-means	Ailon et. al. 2009	2.6.1

Tabla 3.2. Resumen de algoritmos utilizados en Speed Layer

Para la representación del documento en la capa speed se utiliza un algoritmo conocido como TF-ICF descrito en (Reed et al., 2006) que analizaremos en detalle en 3.4.3. Durante el procesamiento de los documentos se ha utilizado una técnica para representar los vectores de rasgos de un documento de forma compacta conocida como "hashing de rasgos" (Detalles en Anexo I).

Finalmente, en los algoritmos de clustering, se ha utilizado distancia coseno porque se ha demostrado más exitosa que la distancia euclídea en dominios dispersos y con elevada dimensionalidad (Strehl et al., 2000). La distancia coseno se define como:

$$distanciaCos(x,y) = 1 - \frac{x \cdot y}{\|x\|_2 \cdot \|y\|_2} = \\ = 1 - \frac{x \cdot y}{\sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2} \cdot \sqrt{{y_1}^2 + {y_2}^2 + \cdots + {y_n}^2}}$$

Para ello se ha desarrollado una modificación en la clase scala del algoritmo de k-means de Spark para utilizar distancia coseno, ya que la implementación actual solamente maneja distancia euclídea. La modificación ha consistido en sustituir en el algoritmo presentado en 2.5.1 la distancia euclídea por la distancia coseno[10].

3.3 Conjunto de datos maestro

La representación de los datos en el conjunto de datos maestro (master dataset) es sencilla pero contiene toda la información necesaria para reconstruir todos los cálculos. Se trata de un conjunto de documentos englobado en un corpus, y cada documento se compone de un conjunto de términos o palabras que conforman el documento.

En la Figura 3.2 se representa un diagrama UML de clases para representar las entidades implicadas en el conjunto de datos maestro. Los documentos poseen un identificador único que permite identificarlos posteriormente en los resultados.

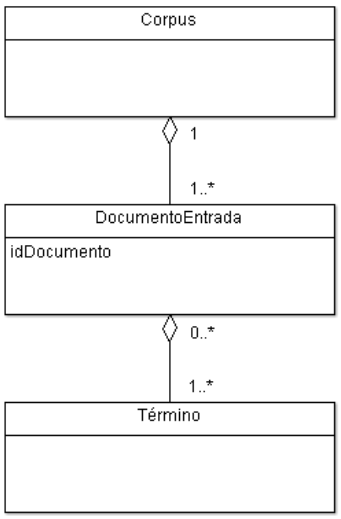

Figura 3.2. Diagrama UML de entidades del conjunto de datos maestro (master dataset)

[10] Esta modificación es una aportación, en forma de implementación, a la comunidad de software libre y el código fuente se encuentra disponible en el repositorio público en:
https://bitbucket.org/azeotropo/uned-ia-master/

3.4 Arquitectura Lambda

3.4.1 Batch Layer

La ejecución de la capa batch se corresponde con una ejecución cíclica sobre el conjunto de datos completo que contiene todo el corpus. Esto es un proceso por lotes que, a partir de los datos del juego de datos maestro realiza los siguientes pasos:

- Cálculo de TF-IDF para representar según el Modelo de Espacio Vectorial.
- Reducción de dimensionalidad mediante Latent Dirichlet Allocation (LDA).
- Clustering mediante k-means||.

La Figura 3.3 muestra los pasos de la capa por lotes (Batch Layer):

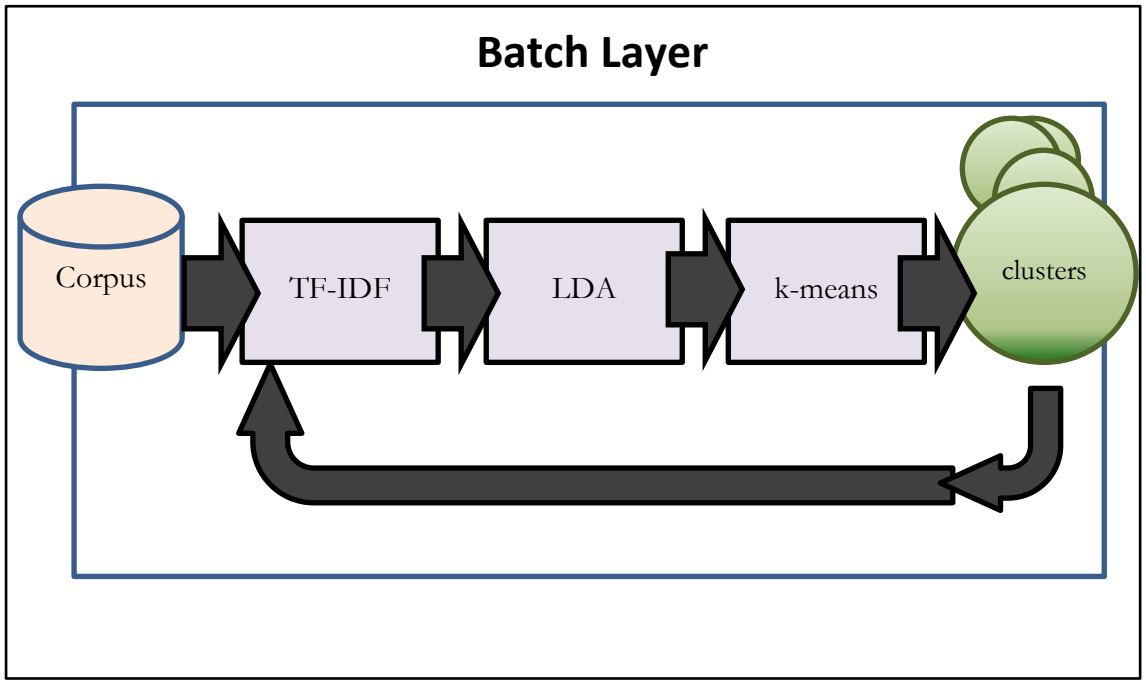

Figura 3.3. Diagrama que representa la implementación de Batch Layer en el sistema diseñado

A continuación se muestran los pasos en pseudocódigo que conforman la capa batch:

1. do
2. $\quad datos \leftarrow cargarDatos()$
3. $\quad \{datosVSM, tablaIDF\} \leftarrow calcularTFIDF(datos)$
4. $\quad \{datosDimReducida, ModeloLenguaje\} \leftarrow calcularLDA(\text{datosVSM})$
5. $\quad clusters \leftarrow kmeans||(datosDimReducida)$
6. $\quad guardarDatos(clusters)$
7. $while\ not(stop)$

Figura 3.4. Pseudocódigo de la ejecución de Batch Layer

La salida de esta capa contiene la información del clustering, que consiste en los centroides de los clusters así como la relación de los documentos asignados a cada cluster. Además se generan dos tablas de datos (tablaIDF y modeloLenguaje) que serán de utilidad en la capa speed. La tablaIDF relaciona cada término con su frecuencia inversa de

documento. El modelo del lenguaje recoge las relaciones que existen entre los documentos y las temáticas así como entre los términos y las temáticas. En la capa speed se utilizará solamente la relación entre términos y temáticas.

3.4.2 Speed Layer

La ejecución de la capa speed se corresponde con un proceso análogo al explicado en la sección anterior pero trabajando sobre un flujo. Aunque en ambas capas se debe realizar el mismo procesamiento, en el caso de la capa batch se trabaja con todo el corpus, mientras que en la capa speed se realiza sobre cada nuevo documento que se debe ingestar.

Por tanto en la capa speed estos son los pasos que deben ejecutar para cada nuevo documento (Figura 3.5):

- Estimación de TF-IDF para representar según el Modelo de Espacio Vectorial.
- Reducción de dimensionalidad mediante datos de LDA.
- Clustering mediante streaming k-means.

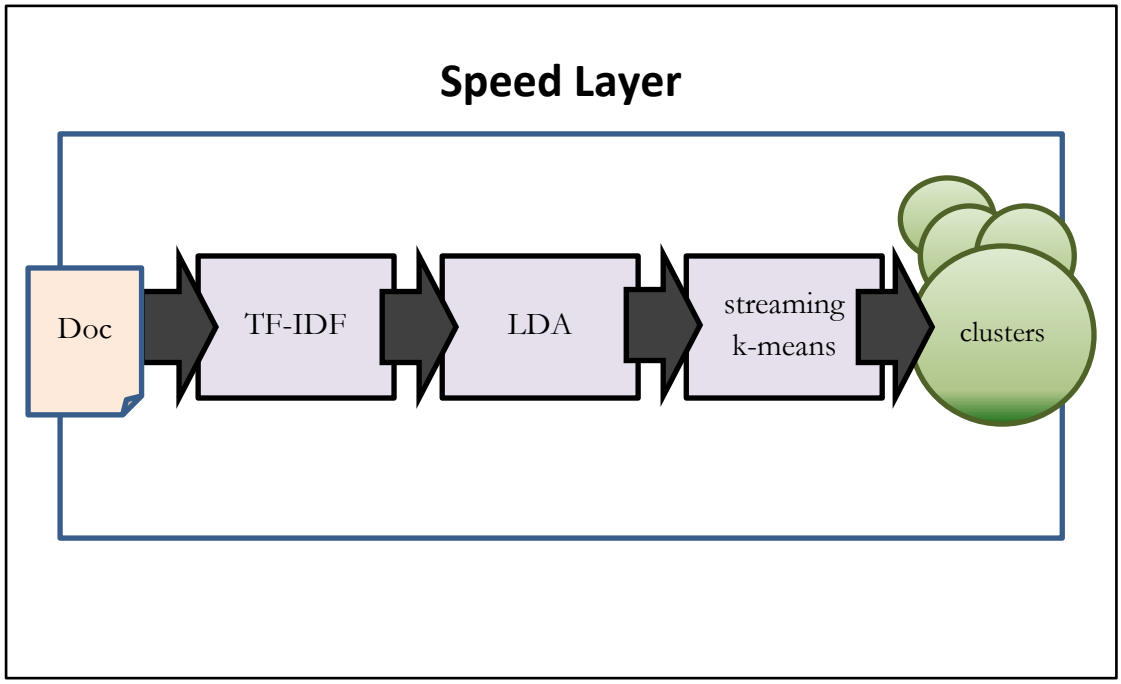

Figura 3.5. Diagrama que representa la implementación de Speed Layer en el sistema diseñado

Sin embargo para el cálculo de TF-IDF sobre flujos dinámicos de documentos como es el de esta capa, no se dispone de la información necesaria para calcular la frecuencia inversa del documento de los términos, por lo que se ha realizado la estimación mediante la técnica TF-ICF (Reed et al., 2006) teniendo en cuenta la información del corpus disponible en la última ejecución de la capa batch. La técnica TF-ICF se basa en precalcular una tabla con la frecuencia inversa del documento para cada término del corpus, de forma que se puede calcular el valor de TF-IDF en flujos de documentos mediante una búsqueda en dicha tabla.

Es decir, se calcula una tabla a partir del corpus:

$$tablaIDF(term) = \log\left(\frac{|D|+1}{1+(d \in D: t \in d)}\right)$$

Que permite calcular el valor de TF-IDF para cada término en un documento del flujo:

$$tfidf(term)_{doc} = (tf(term)_{doc}) * tablaIDF(term)$$

(Cuando no existe un término en el corpus se multiplica por el factor: $\log\left(\frac{|D|+1}{1}\right)$)

Algo parecido ocurre con la reducción de dimensionalidad en la capa speed, que no se dispone de información de todo el corpus. Para realizar la estimación se utiliza el modelo del lenguaje obtenido en la última ejecución de la capa batch.

El modelo del lenguaje es una tabla que registra la relación de un término con sus temáticas, que se obtuvo a partir del corpus en la fase de evaluación de LDA de la capa batch.

Es posible buscar en la tabla el valor para un término:

$$modeloLenguaje(termino) = \{tematica_1, tematica_2, \dots, tematica_n\}$$

Para realizar la reducción de la dimensionalidad de un documento, se utiliza la tabla con la información del modelo del lenguaje:

$$reducciónDim(doc) = reducciónDim([term_1, term_2, .. term_n]) =$$
$$normalizacion(\sum_{tematica} \begin{bmatrix} peso_{TFIDF}(term_1) \cdot modeloLenguaje(term_1) \\ \dots \\ peso_{TFIDF}(term_n) \cdot modeloLenguaje(term_n) \end{bmatrix}, 1)$$

Esto permite estimar una reducción de la dimensionalidad a través de las temáticas de los términos que componen el documento.

Finalmente, en el algoritmo de clustering de la fase speed, se inicializan los centroides antes del proceso de streaming con los centroides calculados en la última fase batch, para que, al menos inicialmente, exista una correspondencia entre los clusters de ambas capas.

A continuación (Figura 3.6) se muestran los pasos en pseudocódigo que conforman la capa speed. La entrada consiste en un flujo de documentos que se incorporan en el tiempo. Los otros tres datos de la entrada proceden de la última ejecución en la capa batch.

Entrada:

- $Streaming(t)$: $doc_1, doc_2, \ldots, doc_n$
- $datosIDF$
- $modeloLenguaje$
- $clustersIniciales$

Pseudocódigo que conforma la capa speed:

1) $streamingKmeansInit(clusterIniciales)$
2) $foreach\ nuevoDoc\ in\ lee(Streaming)\ do$
3) $\quad docVSM \leftarrow estimarTFICF(nuevoDoc, datosIDF)$
4) $\quad datosDimReducida \leftarrow estimarLDA(docVSM, modeloLenguaje)$
5) $\quad clusters \leftarrow streamingKmeans(datosDimReducida)$
6) $\quad guardarDatos(clusters)$
7) $end\ foreach$

Figura 3.6. Pseudocódigo de la ejecución de Speed Layer

La salida de este algoritmo son los centroides de los clusters así como la relación de cada documento con cada cluster. Es importante resaltar que la salida de esta capa está disponible inmediatamente para cada documento nuevo, por lo que la evolución de los centroides de los clusters se registra en tiempo real.

La información generada por ambas capas queda registrada y disponible bajo consulta desde la capa serving. Es el consumidor de los datos el que puede decidir si quiere recuperar los datos de la capa batch y speed de forma conjunta o separada mediante consultas ad hoc, ya que dispone de campos para discriminar por el origen de los datos.

3.4.3 Serving Layer

La base de datos HIVE, utilizada en la capa proveedora o Serving, permite configurar como fuente de datos, directamente los ficheros de salida en formato texto procedentes de las capas que procesan datos, y consultarlos como tablas SQL (Figura 3.8).

El modelo de datos depende en gran medida de las necesidades de la aplicación final. En este trabajo se ha diseñado el modelo de datos pensando en cumplir el requisito de modelar la agrupación de documentos similares en clusters, aunque podría añadirse información disponible obtenida a través de LDA como la información de temáticas (topics).

El modelo de datos recoge la relación de agrupación entre los clusters y los documentos que ha sido descubierta por las técnicas de minería de textos.

En la Figura 3.7 se muestra el diagrama UML de las entidades presentes en el modelo de datos que básicamente contienen la información sobre el cluster y los documentos además de la relación entre ambos.

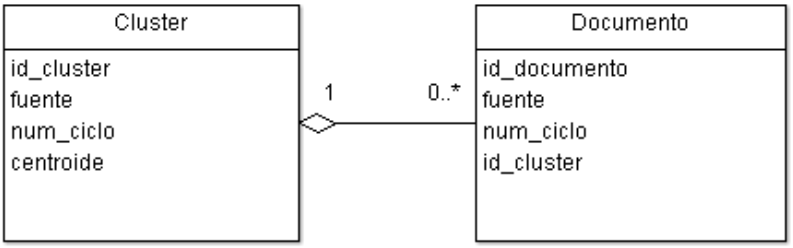

Figura 3.7. Diagrama UML de entidades en Serving Layer

Es de destacar el campo "fuente" que permite diferenciar el origen (speed o batch) del cluster o de la asociación de un documento con un cluster. De esta forma, desde la capa serving, el consumidor de los datos puede obtener los resultados a través de una consulta SQL opcionalmente discriminando por fuente. De esta forma se consigue flexibilidad debido a que es posible consultar los documentos y su relación con su cluster, bien sea de forma separada, diferenciando los generados por la capa batch o speed, o bien de forma conjunta.

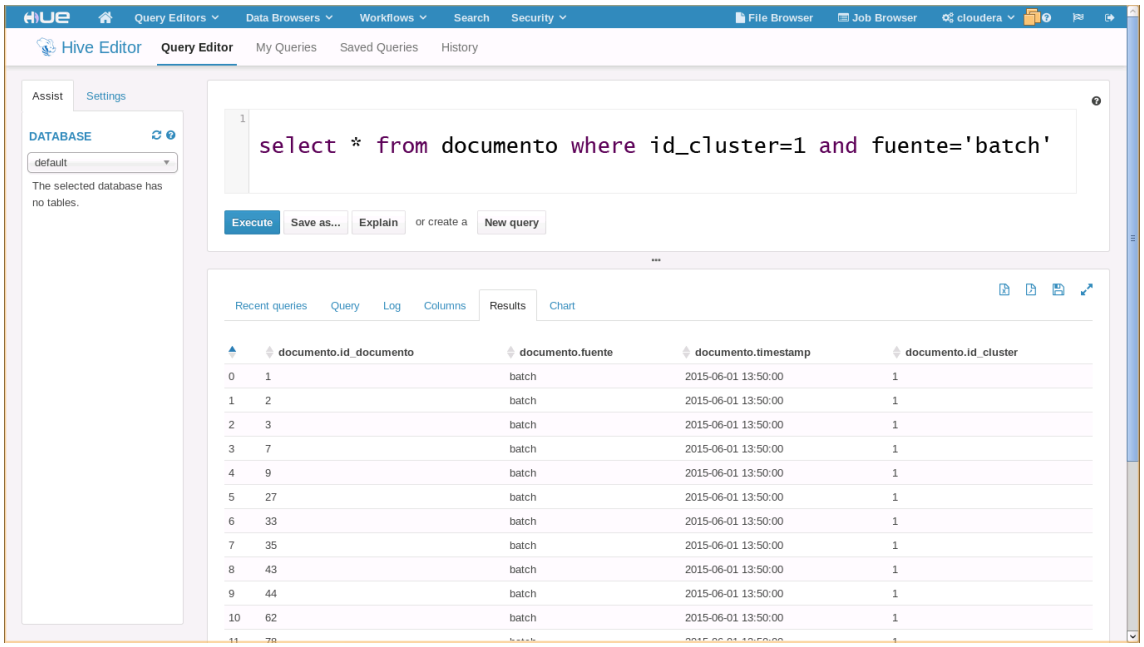

Figura 3.8. Ejecución de consulta SQL en HIVE sobre los documentos

4 Experimentos y resultados

En este capítulo se describe el diseño experimental realizado en este trabajo para obtener medidas de calidad y escalabilidad. También se muestran los resultados obtenidos y se analizan para contrastar con las cuestiones planteadas en los objetivos.

4.1 Juegos de datos

En este trabajo se han utilizado juegos de datos de MoviLens[11] y Reuters-21578[12].

MovieLens

El conjunto de datos MovieLens 100k, recopilado por la Universidad de Minnesota a través del proyecto GroupLens Research Project, consiste en 100.000 puntuaciones de 943 usuarios y 1682 películas. Un usuario correspondería a un documento y las películas que ha puntuado corresponderían a las palabras. Las películas pertenecen a 19 géneros que corresponderían a las temáticas. Cada usuario ha puntuado al menos 20 películas. En otros trabajos se ha utilizado porque no requiere preprocesamiento (Zhao et al., 2014)(Wang et al., 2002) (Jin et al., 2011) y en este trabajo se han utilizado para poder comparar con los resultados de escalabilidad presentes en esos trabajos.

Reuters-21578

Por otra parte, se ha utilizado el juego de datos "Reuters-21578 Distribution 1.0" que es un juego de datos con información textual muy popular para experimentos de minería de textos. Contiene 21578 documentos de noticias de Reuters desde 1987 etiquetados de forma manual por personal de Reuters. Se encuentra etiquetado mediante 5 clases de categorías (people, places, topics, orgs, exchanges) con valores de 672 categorías pero algunas pueden ocurrir muy raramente. Algunos documentos pertenecen a muchas categorías y otros solamente a una. Los documentos se encuentran etiquetados en formato SGML. El corpus en total ocupa 27 MB.

4.2 Experimentos

Se han realizado los experimentos sobre un equipo Intel Core i5 3.20GHz con 4 cores y 16Gb de RAM. Se han ejecutado sobre una máquina virtual en VirtualBox. Se ha utilizado un sistema operativo huésped Linux (Lubuntu 64bits) donde se configuraban las restricciones de CPU y memoria: la configuración de CPU ha sido 1, 2, y 4 cores mientras que la configuración de memoria se ha establecido en 12 Gb.

Se ha ejecutado Apache Spark (versión 1.3.1) en modo standalone ejecutando el multiproceso mediante los distintos cores disponibles en la máquina. En el desarrollo se ha utilizado Java 8 y Scala 2.10.4.

Se ha estudiado la respuesta del sistema construido frente a la escalabilidad y la calidad ya que es uno de los objetivos fijados. Para facilitar su estudio se han realizado los

[11] http://grouplens.org/datasets/movielens/
[12] http://www.daviddlewis.com/resources/testcollections/reuters21578/

experimentos de la capa batch y speed por separado, principalmente por las limitaciones de recursos de la máquina de experimentos. Se han tomado tiempos de ejecución globales y de cada paso implicado para analizar con otros trabajos.

Para evaluar la escalabilidad se ha utilizado el juego de datos de MovieLens. Se ha replicado este conjunto de datos para conseguir un tamaño adecuado para las pruebas. Para ello se han realizado réplicas de 32, 64, 128 y 256 veces la colección original. Esta última tiene un tamaño de 8Gb. Se ha diseñado de esta forma para poder comparar los resultados con experimentos similares.

En los experimentos se ha realizado reducción de dimensionalidad y clustering para un número de dimensiones y de clusters fijado en 19 que corresponde a las temáticas ocultas.

Para evaluar la calidad del clustering se ha utilizado el juego de datos Reuters-21578 porque MovieLens no era adecuado para este propósito ya que no se encuentra etiquetado. Sobre estos documentos se han eliminado las palabras vacías del inglés. Se han seleccionado únicamente documentos de las 8 categorías mayoritarias que estuviesen categorizados con una única categoría (Tabla 4.1). Se ha realizado de esta manera ya que los algoritmos de clustering utilizados sólo asignan cada documento a un único grupo.

En estos experimentos se ha fijado el número de clusters en 8, tantos como categorías, y se han realizado experimentos para evaluar la calidad en función del número de dimensiones destino de la reducción de dimensionalidad.

Categoría
earn
acq
grain
crude
trade
money-fx
interest
ship

Tabla 4.1. Categorías de Reuters de los documentos utilizados en evaluación de calidad

4.3 Estudio de la calidad

Uno de los objetivos de este trabajo es estudiar si al aplicar la arquitectura lambda es posible obtener resultados de clustering sobre Big Data en tiempo real y se siguen obteniendo calidades de clustering comparables a las que se obtienen en otros trabajos. Por ello se han realizado muestreos en diferentes configuraciones para evaluar la calidad. Una optimización de la calidad a partir de los parámetros queda fuera del objetivo de este trabajo.

4.3.1 Análisis de Batch Layer

Para realizar los experimentos se han evaluado varias configuraciones y se han repetido los experimentos tres veces con cada configuración para poder disponer de valores promedio de las medidas de calidad. Se ha configurado el número de clusters de

k-means como el número de clases presentes en la colección de datos (k=8) y se ha reducido la dimensionalidad en LDA a varios valores de dimensiones (16, 32, 50, 64, 80 100, 128, 256, 512, 1024) y en el paso de TF-IDF se han probado configuraciones que descartaban en el cálculo aquellos términos que no estuviesen presentes en al menos un número de documentos (1, 2, 3, 4, 5 y 6).

En la Tabla 4.2 se muestran los mejores resultados obtenidos en los experimentos. Se muestra el mejor rendimiento obtenido, el promedio de los resultados obtenidos con esa configuración y la configuración que en promedio obtiene los mejores resultados. La configuración que obtuvo el mejor resultado de forma aislada (medida-F de 0.72) configurada con una reducción a 256 dimensiones en la fase de LDA y en la fase de TF-IDF se descartó aquellos términos que no estuviesen presentes al menos en 6 documentos.

La configuración que obtuvo un mejor promedio de medida-F con un valor de 0.57 utilizó 80 dimensiones en LDA y descartó en la fase TF-IDF aquellos términos que no estuviesen en al menos 3 documentos.

Experimento/Métrica	Medida-F	NMI	Pureza
Mejor resultado individual (n_{LDA}=256) (minfrec=6) Número de clusters: k=8	0.7209289	0.5280520	0.7876286
Promedio con el mejor resultado individual (n_{LDA} =256) (minfrec=6) Número de clusters: k=8	0.55746489	0.40474370	0.60643974
Mejor promedio (n_{LDA} =80) (minfrec=3) Número de clusters: k=8	0.57429448	0.40338974	0.65455777

Tabla 4.2. Resultados de calidad obtenidos en los experimentos sobre Batch Layer

En la Tabla 4.3 se muestran los resultados de calidad que se han obtenido en experimentos análogos de clustering con k-means sobre el corpus de Reuters y vemos que los resultados obtenidos en este trabajo son comparables o superiores a los resultados presentes en otros trabajos del estado del arte. Las medidas de calidad reflejadas en la tabla son medida-F y pureza que son las utilizadas en esos trabajos.

No se ha utilizado exactamente el mismo conjunto de documentos ya que los trabajos no precisan el conjunto de documentos concreto. En cualquier caso, los resultados pueden ser comparables ya que se trata de subconjuntos del corpus Reuters.

Referencia	Calidad
Larsen & Aone, 1999 **Corpus: Reuters** **Clustering: k-means**	Medida-F=0.49-0.53
Steinbach et al., 2000 **Corpus: Reuters** **Clustering: Bisecting k-means**	Medida-F=0.5863-0.7067
Beil et al., 2002 **Corpus: Reuters** **Representación: TF** **Clustering: Bisecting k-means, 9-Secting k-means, HFTC**	Medida-F=0.43-0.57
Hotho et al., 2003 **Corpus: Reuters** **Representación: TF-IDF** **Clustering: Bisecting k-means**	Pureza=0.461-0.57

Tabla 4.3. Referencias de medidas de calidad en clustering sobre el corpus Reuters

4.3.2 Análisis de Speed Layer

Los experimentos para evaluar la calidad en la capa speed se han realizado sobre un flujo de documentos de 2000 documentos. Se han realizado experimentos con varias configuraciones (TF-IDF frecuencia mínima de documento: 5-10. LDA: 16 y 32 dimensiones) y en la Tabla 4.4 se muestran los mejores resultados.

Experimento	Medida-F	NMI	Pureza
Mejor resultado en medida-F **n_{LDA}=16** **min-$frec_{TFIDF}$=10** **Número de clusters: k=8**	0.69346535	0.5638645	0.81088436
Mejor resultado en NMI **n_{LDA}=16** **min-$frec_{TFIDF}$ =10** **Número de clusters: k=8**	0.6685507	0.5850638	0.8121212

Tabla 4.4. Resultados de calidad obtenidos en los experimentos sobre Speed Layer

En la Tabla 4.5 se muestran los resultados de calidad obtenidos en clustering textual en streaming (Zhong, 2005).

Referencia	Calidad
Zhong, 2005 **Corpus: NG20 (Textual)** **Clustering: OSKM (online spherical k-means)** **2000 documentos** **Número de clusters: k=5-20**	NMI= 0.4-0.5

Tabla 4.5. Referencias de medidas de calidad en clustering textual en streaming

Al procesar documentos en streaming en la capa speed se obtiene un valor de Información Mutua Normalizada (NMI) de 0.58 que es un resultado comparable a los obtenidos en otros trabajos similares.

4.3.3 Conclusiones

La calidad del clustering que se consigue en esta propuesta tanto en la capa batch como en la capa speed es comparable o ligeramente superior a los obtenidos en otros trabajos no orientados a Big Data, pero que realizan tareas similares, es decir, clustering sobre documentos de texto. Se observa en la capa speed que a pesar de las aproximaciones tomadas se obtienen buenos resultados de calidad.

El buen resultado de calidad de la capa speed puede explicarse porque la inicialización de centroides en esta capa no es aleatoria sino que proviene de la capa batch. Los algoritmos de la capa batch han calculado un conjunto de centroides dispersos y adecuados para inferir buenas agrupaciones de documentos.

4.4 Estudio de la escalabilidad

4.4.1 Análisis de Batch Layer

Los experimentos para estudiar la escalabilidad se han realizado con 32, 64, 128 y 256 veces la colección original de MoviLens para poder obtener medidas y compararlas con trabajos que realizan experimentos con el mismo tamaño (Zhao et al., 2014). El conjunto de datos con 256 veces el juego de datos original tiene un tamaño de unos 8Gb. Cada experimento se ha repetido tres veces para verificar empíricamente su reproducibilidad.

Se han tomado tiempos del proceso global y de los pasos intermedios para realizar un análisis a nivel general y de las partes.

A continuación se muestran las gráficas de scaleup global y por pasos (Figura 4.1) y speedup del proceso global (Figura 4.2).

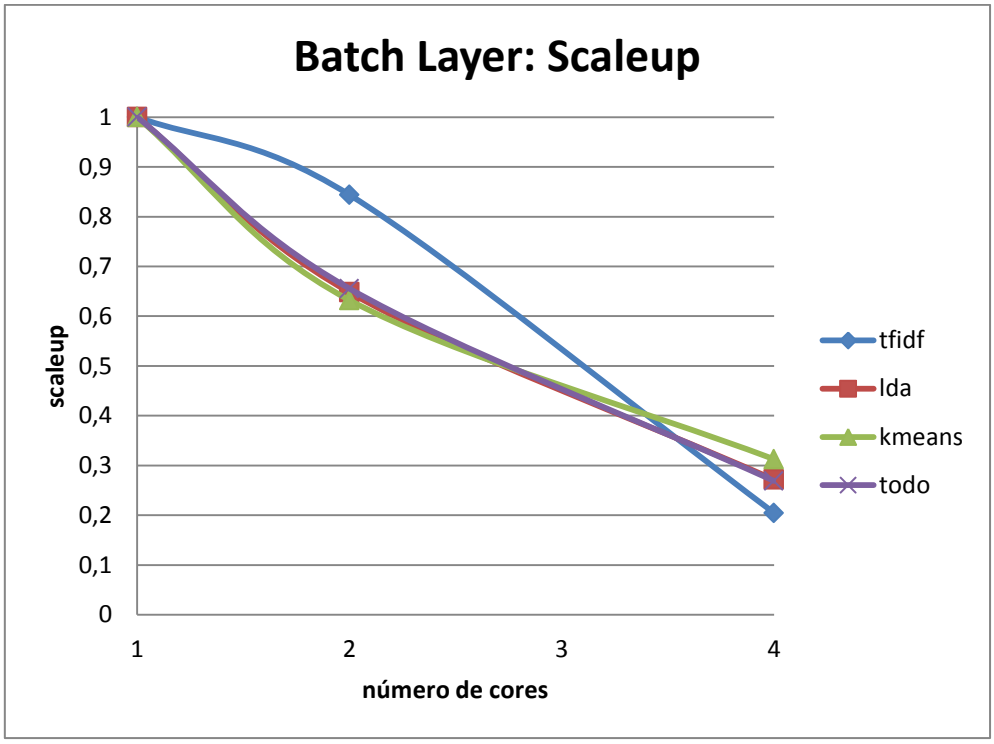

Figura 4.1. Scaleup en Batch Layer para el proceso global y diferentes pasos

La (Figura 4.1) muestra valores descendientes de scaleup al aumentar el número de cores. De todos los pasos, podemos ver que el que mejor escala es k-means, que para cuatro cores obtiene un valor de scaleup de 0.33, mientras que el que peor escala es el paso de representación (TF-IDF) que se queda en 0.20, aunque todos los pasos siguen la misma tendencia. El proceso en conjunto queda con un valor de 0.27 que implica una escalabilidad escasa.

El hecho de que todos los pasos sigan la misma tendencia indica que no existe un cuello de botella individual que podría indicar un error aislado.

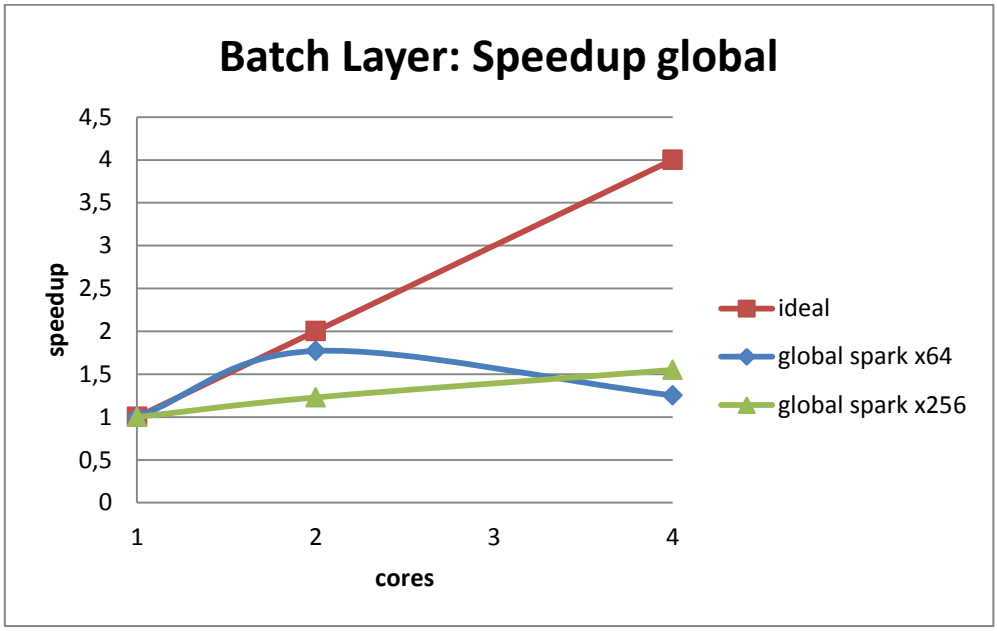

Figura 4.2. Speedup en Batch Layer para el proceso global

La (Figura 4.2) muestra la evaluación de speedup para la ejecución sobre el juego de datos con 64 y 256 veces la colección original. Con dos cores se llegan a valores de 1.77 que corresponde a una eficiencia del 88.5%. Para speedup con cuatro cores, se observan en ambas casos valores cercanos a 1.5, lo cual indica que añadiendo más cores el proceso se hace menos eficiente (eficiencia 37.5%).

Se observa que la escalabilidad se degrada aumentando la capacidad de computación mediante incremento del número de cores, pero de la misma forma incluso en colecciones de distinto tamaño. Si se hubiese presentado un peor speedup para la colección mayor podría indicar un punto de saturación o degradación del rendimiento por ejemplo debido a una configuración errónea que obligase a realizar una excesiva paginación en disco (swapping). Por tanto se descarta que la degradación de escalabilidad se deba a que se alcanza un punto saturación.

La Figura 4.3 muestra la evolución de speedup sobre la operación de clustering para el conjunto de datos 64 veces y 256 veces la colección original. Tomando los valores de cuatro cores podemos observar que sobre la colección mayor obtiene mejor speedup que corresponde a una eficiencia del 42%. Se pueden aplicar las mismas conclusiones que sobre el speedup global, es decir se observa que se degrada la escalabilidad del clustering pero se descarta que se alcance un punto de saturación en esta fase.

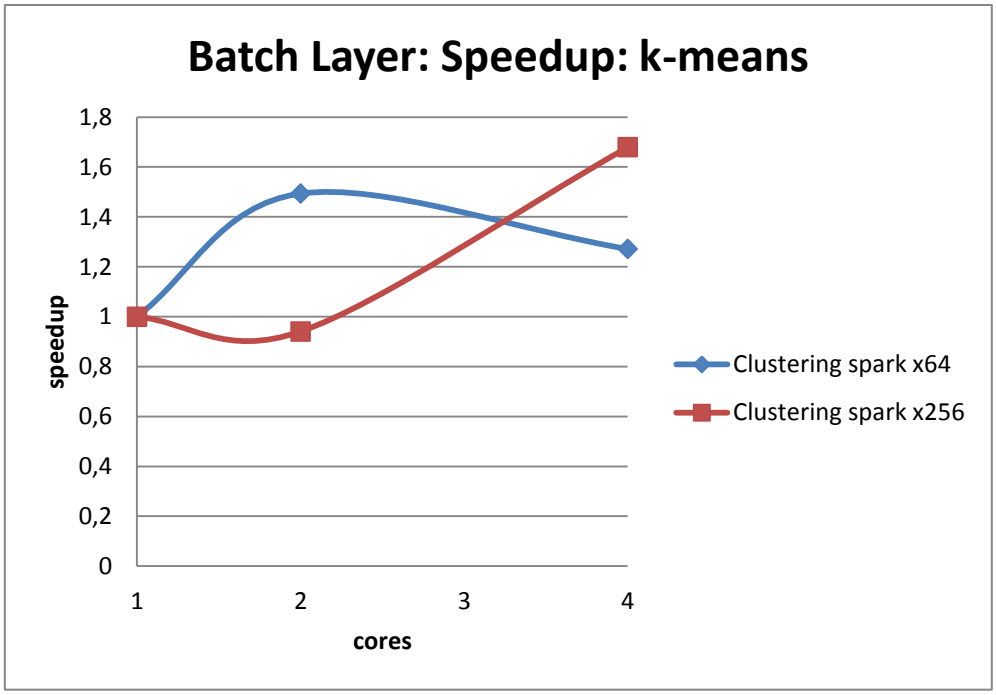

Figura 4.3. Speedup en Batch Layer de la fase de clustering

La Figura 4.4 muestra la evolución de la magnitud speedup sobre la operación de LDA para los conjuntos de datos 64 y 256 veces la colección original. Se observan en los valores obtenidos para cuatro cores que a pesar de agregar más cores no está mostrando capacidad de adaptarse al crecimiento de los datos. Las conclusiones son similares a las de speedup global, a saber, se degrada la escalabilidad de LDA pero se descarta que se deba a que se alcanza un punto de saturación en esta fase.

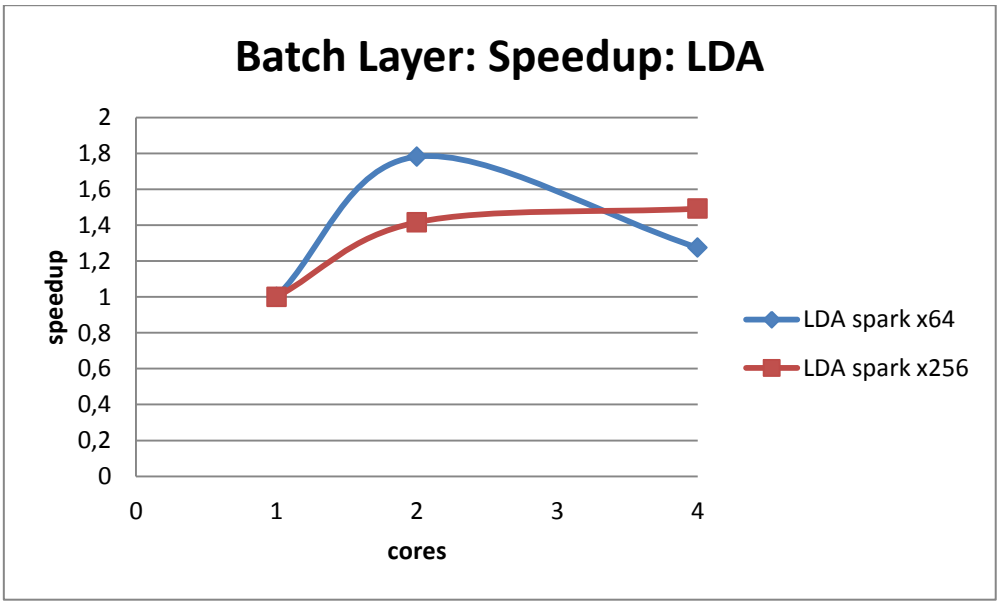

Figura 4.4. Speedup en Batch Layer de LDA

Las gráficas a continuación (Figura 4.5, Figura 4.6 y Figura 4.7) muestran la evolución de la escalabilidad (scaleup y speedup) de LDA evaluado con el API de Spark frente a otros trabajos que han realizado reducción de dimensionalidad con el mismo juego de datos. Podemos concluir que los resultados obtenidos en este trabajo son inferiores en

cuanto a escalabilidad frente a trabajos similares (Zhao et al., 2014) (Li et al., 2012). La diferencia se advierte especialmente en los resultados obtenidos con 4 cores.

Los valores de escalabilidad obtenidos son peores de lo esperado, ya que se esperaba que Spark se aprovechase de sus ventajas a la hora de ejecutarse en este procesamiento iterativo frente a la rigidez de los flujos de datos fijos en Hadoop.

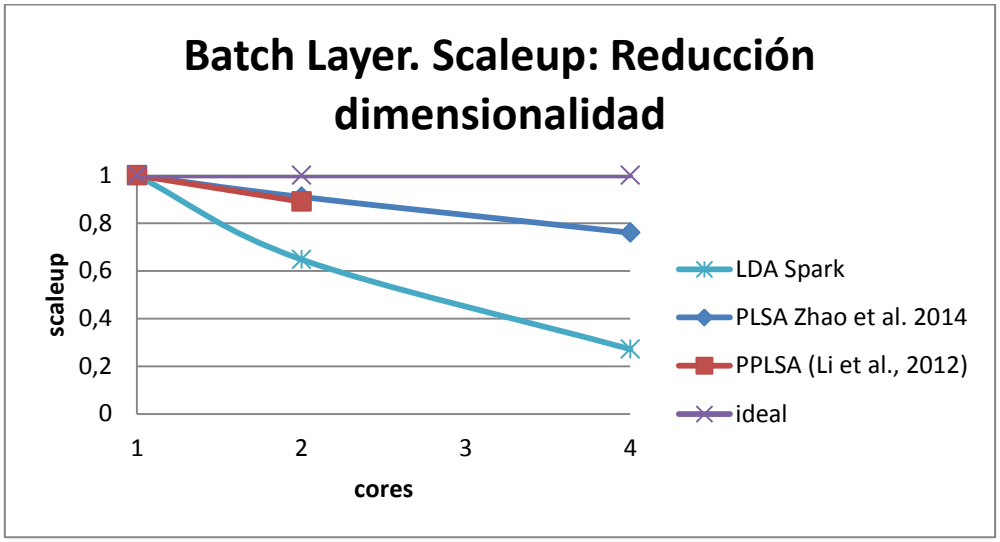

Figura 4.5. Scaleup en Batch Layer de LDA comparada con (Zhao et al., 2014) y PPLSA (Li et al., 2012)

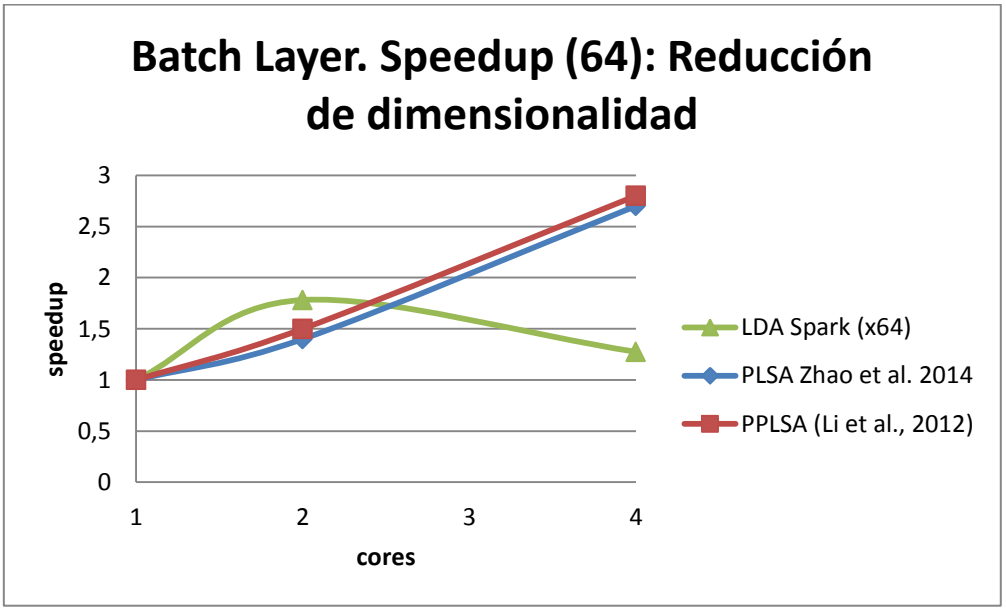

Figura 4.6. Speedup en Batch Layer de LDA comparada con (Zhao et al., 2014) y PPLSA (Li et al., 2012) sobre el conjunto de datos x64.

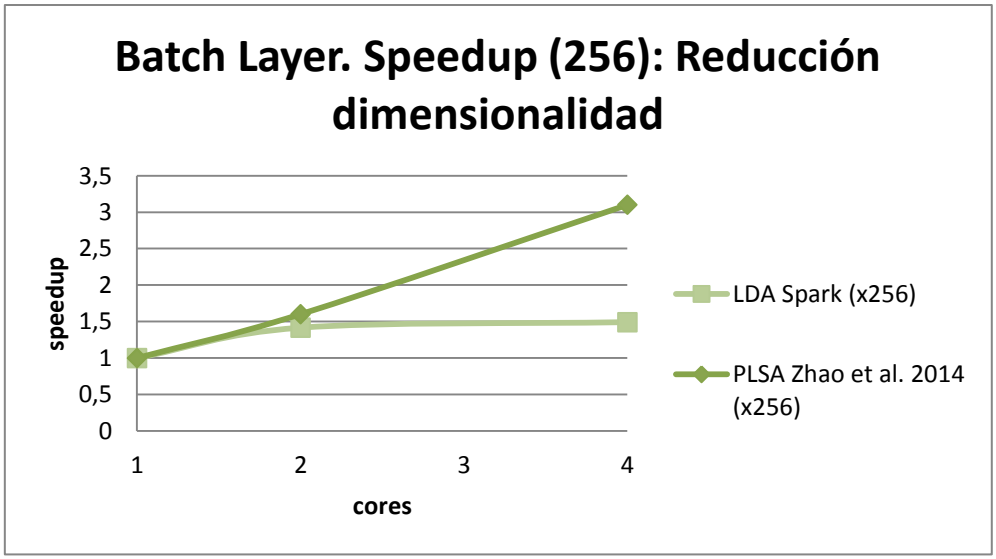

Figura 4.7. Speedup en Bath Layer de LDA comparada con (Zhao et al., 2014) sobre el conjunto de datos x256

Las figuras siguientes (Figura 4.8, Figura 4.9, Figura 4.10) muestran la evolución de la escalabilidad (scaleup y speedup) del paso de clustering realizado en la capa batch con el API de Spark comparado con (Zhao et al., 2009). Se observan en este trabajo valores de escalabilidad inferiores a (Zhao et al., 2009). Al igual que en el paso anterior se observan mayores diferencias en el caso de 4 cores.

Al igual que en el punto anterior, aquí también se esperaba que Spark, en este procesamiento iterativo, rindiese mejores resultados de escalabilidad. Cabía esperar un mejor rendimiento frente a las implementaciones en Hadoop, ya que estas últimas utilizan un flujo de datos fijo que necesita escribir los datos a disco (HDFS) para ejecutar la siguiente iteración.

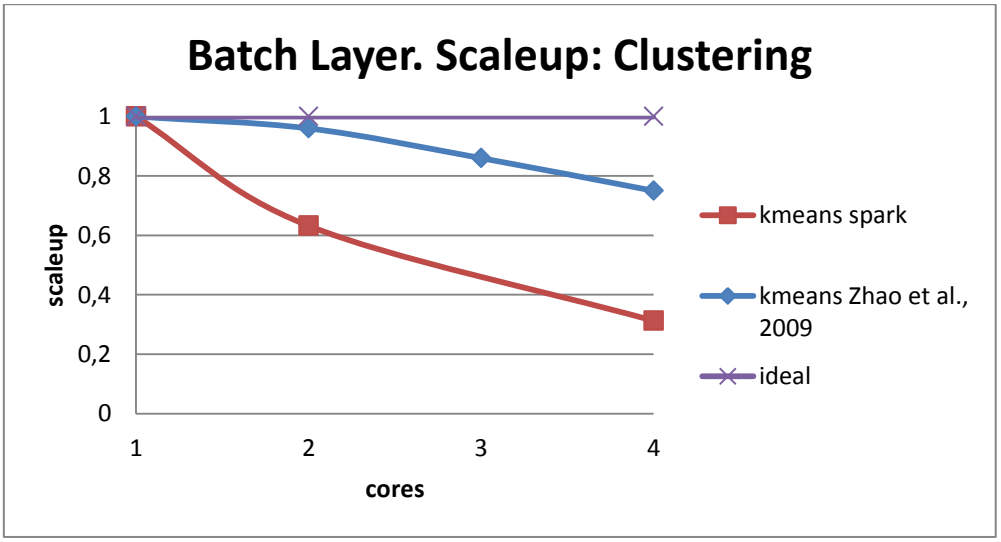

Figura 4.8. Scaleup en Batch Layer de clustering comparada con (Zhao et al., 2009)

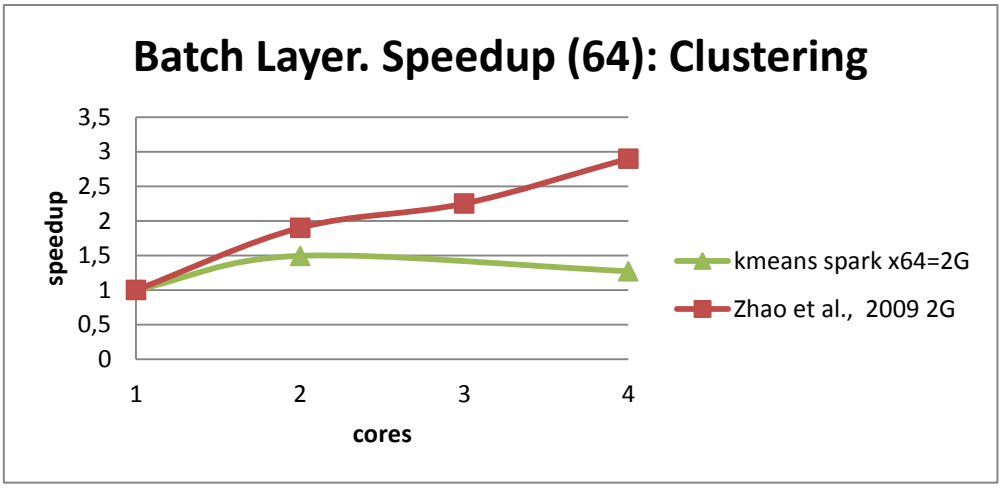

Figura 4.9. Speedup en Batch Layer de clustering comparada con (Zhao et al, 2009) con conjunto de datos equivalente (2Gb)

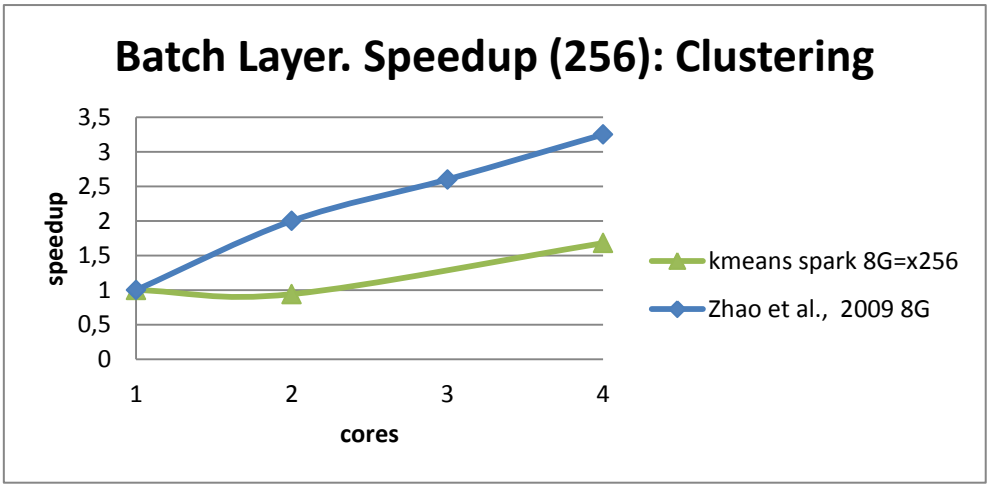

Figura 4.10. Speedup en Batch Layer de clustering comparada con (Zhao et al, 2009) con un conjunto de datos equivalente (8Gb)

4.4.1.1 Interpretación

La escalabilidad de esta capa en este trabajo es bastante inferior a la comparada frente a otros sistemas que trabajan con datos masivos en operaciones similares. A través del análisis de las gráficas se descarta que se lleguen a puntos de saturación o que haya cuellos de botella en alguno de los pasos.

Sin embargo hay una diferencia importante que debemos considerar entre los experimentos de este trabajo frente a la metodología de todos los trabajos con los que se han comparado: mientras que los experimentos en otros trabajos se ejecutan directamente sobre un cluster de máquinas en este trabajo se ejecutan sobre la misma máquina y además en una máquina virtual.

El hecho de que sea una misma máquina tiene consecuencias en los resultados obtenidos en este trabajo, que penalizan el rendimiento, debido a la utilización de recursos compartidos como es el disco duro. Cuanto mayor es el número de cores peor es el rendimiento, como se observa experimentalmente, ya que el mismo recurso se tiene que compartir entre más subprocesos.

Adicionalmente el rendimiento se penaliza debido a la utilización de una máquina virtual ya que si la máquina física tiene 4 cores pueden no estar completamente dedicados al huésped si además necesita dedicar una parte a la propia virtualización. De ahí que se observen diferencias mayores para la ejecución en 4 cores.

4.4.2 Análisis de Speed Layer

Para la ejecución de los experimentos de esta capa se ha realizado la entrada de datos en forma de flujo de datos (streaming), que simula la ingesta de nuevos documentos en el sistema. Se ha utilizado la colección MovieLens 100k como datos de entrada.

En esta capa se ha efectuado la evaluación en dos partes: (1) por una la evaluación del proceso global en función del número de documentos procesados (sección 4.4.2.1) y (2) por otra la evaluación de los dos primeros pasos en función del crecimiento del tamaño de las tablas de datos que utilizan (sección 4.4.2.2).

4.4.2.1 Resultados globales

Con el fin de realizar los experimentos en esta capa, se ha evaluado el número de documentos que se pueden procesar en una cantidad de tiempo fija (15, 30 y 60 minutos), y se ha comparado la escalabilidad en su ejecución con 1, 2 y 4 cores. La definición de los intervalos de tiempo se debe a que aproximadamente una hora es el máximo tiempo de ejecución obtenido en los experimentos de la capa batch. El resto de tiempos son valores compatibles con el cálculo de la escalabilidad para el número de cores indicados.

Cada experimento se ha repetido tres veces y se representan los valores promedio. Se ha utilizado un factor de decaimiento $\alpha = 1$ para tener en cuenta todos los datos desde el principio, y no se ha realizado ponderación de los centroides.

A continuación, en la Figura 4.11, se muestra la evolución del número de documentos procesados por unidad de tiempo frente al número de cores.

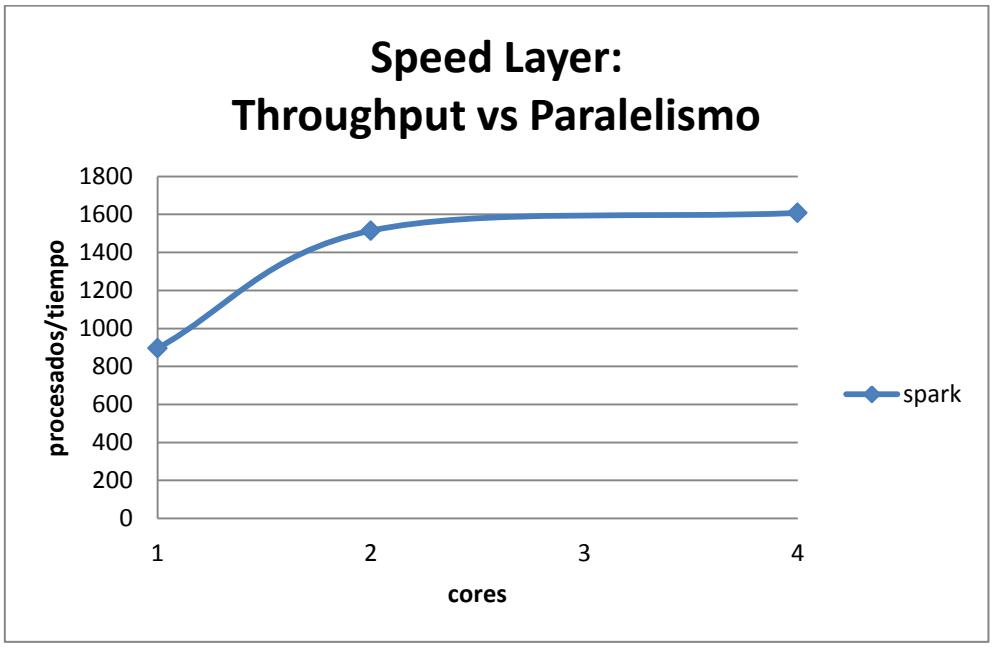

Figura 4.11. Speed Layer. Gráfica de la evolución del throughput frente al número de cores

Se observa que el número de elementos procesados por unidad de tiempo aumenta para 2 cores y luego se estabiliza para 4 cores.

En la Figura 4.12 se muestra la evaluación de scaleup para una ejecución en 60, 30 y 15 minutos, para 4, 2 y 1 cores respectivamente. Se observa para 4 cores un valor de scaleup de 0.55 que supone una escalabilidad superior a la obtenida en la fase batch de 0.27 (Figura 4.1). Esto significa que la capa speed tiene mejor escalabilidad que la capa batch. La explicación de esto es porque la capa speed trabaja sobre conjuntos de datos más pequeños.

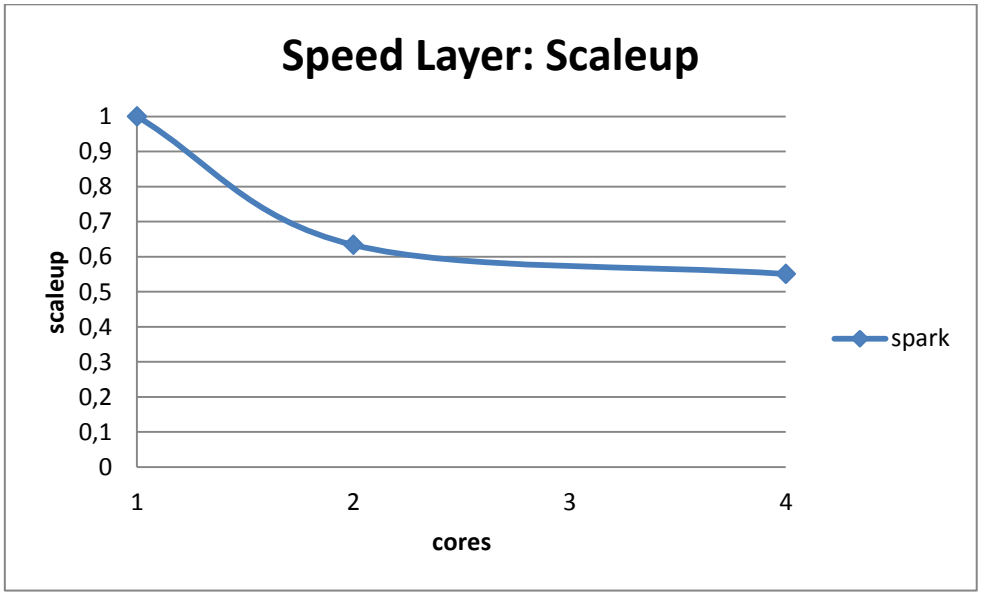

Figura 4.12. Speed layer. Evaluación de Scaleup en la ejecución de streaming

Seguidamente (Figura 4.13) se muestra la evaluación de speedup, para una ejecución en 60 minutos, para distinto número de cores (1, 2 y 4).

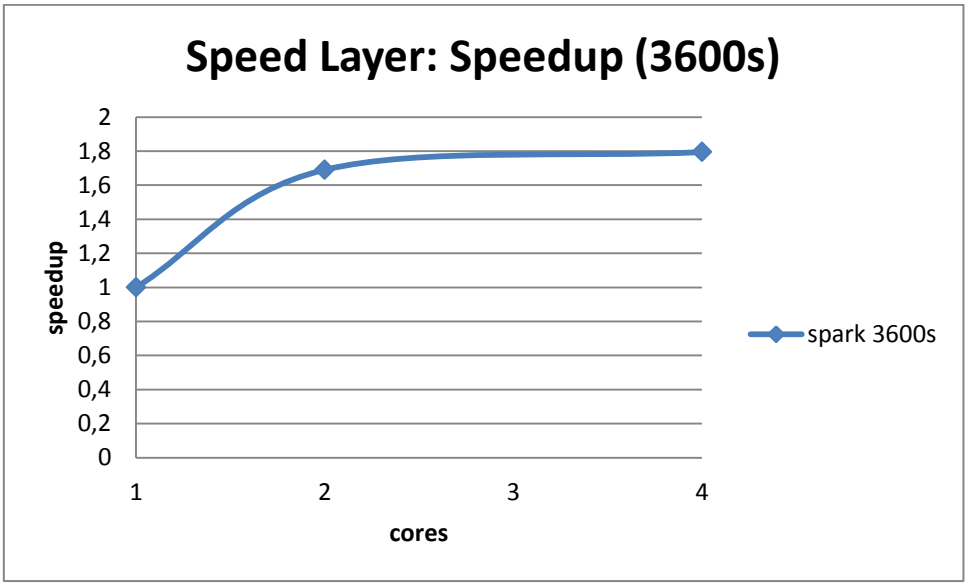

Figura 4.13. Speed layer. Evaluación de speedup en la ejecución de streaming durante una hora

Observando las figuras anteriores (Figura 4.12 y Figura 4.13) vemos que si añadimos más cores al procesamiento, no aumenta proporcionalmente el número de elementos procesados. La eficiencia con dos cores es del 85% mientras que para cuatro

cores es del 45%. Son valores decrecientes de eficiencia. A pesar de esto, se observan valores de escalabilidad superiores a los obtenidos en la fase batch, lo que indica una cierta capacidad para adaptarse a cantidades crecientes de datos. Esta capa experimenta los mismos inconvenientes que se han explicado en la capa batch debidos a la ejecución en una única máquina, por lo que sería de esperar una mejor escalabilidad ejecutándose en un cluster de varias máquinas.

Al analizar esta capa, hay que resaltar la forma en que se procesan los datos, que a diferencia de la capa batch que trabaja sobre todos los datos, en este caso es como una tubería así que la velocidad global será tan lenta como el más lento de sus pasos.

4.4.2.2 Resultados TF-IDF y LDA

En el caso de los dos primeros pasos (TF-IDF, LDA), hay que tener en cuenta para estudiar la escalabilidad frente al tamaño de los datos, que no depende tanto del número de documentos a procesar sino del tamaño de las tablas sobre las que tienen que buscar. Por eso su escalabilidad se ha estudiado aparte en otro experimento, con la misma mecánica que el anterior, pero variando el tamaño de las tablas (con el conjunto de datos completo, la mitad y una cuarta parte) frente a la ejecución con 4, 2 y 1 cores y los resultados se muestran en la Figura 4.14.

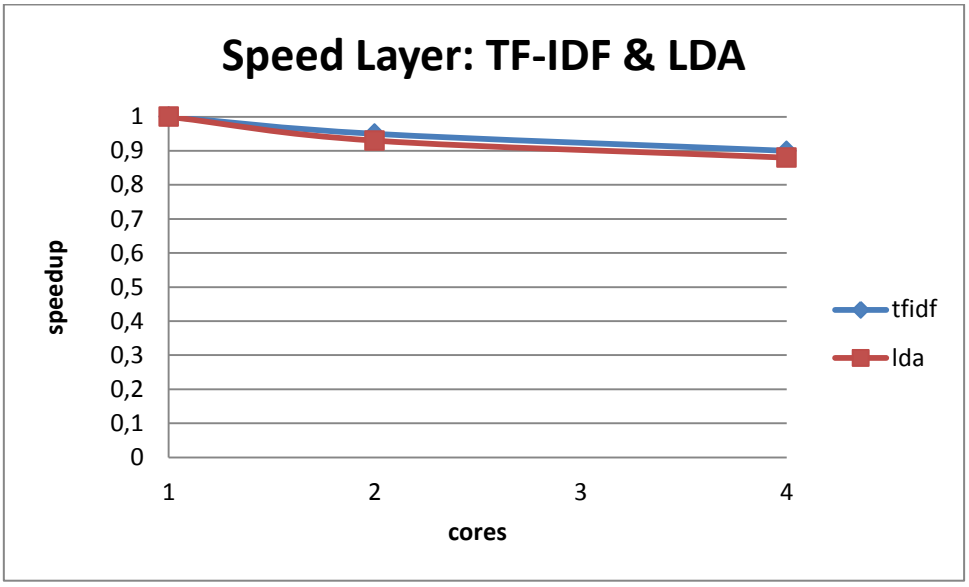

Figura 4.14. Speed Layer. Evolución de scaleup en paso TF-IDF y LDA frente a tamaño de tablas de datos

En ambos casos se observan valores de speedup cercanos a la unidad que es el valor teórico lo cual muestra que la escalabilidad es buena en estos pasos.

En la (Tabla 4.6) se muestra una comparación de tiempos de cada paso del proceso respecto del total. Se observa que aquellas fases que más tiempo consumen son los dos primeros pasos: la representación (TF-ICF) y la reducción de dimensionalidad, lo cual indica que son las porciones preferentes sobre las que realizar optimizaciones.

Paso	Porcentaje de tiempo
TF-ICF	56%
Reducción de dimensionalidad	45%
Clustering k-means	2%

Tabla 4.6. Speed Layer. Porcentaje de tiempo dedicado a cada paso

4.4.2.3 Interpretación

La principal aportación de la capa speed es que es responsable de ofrecer resultados de clustering en contextos bigdata en tiempo real con una calidad notable.

En la sección 4.4.2.1 se ha indicado que al añadir más cores al procesamiento de la capa speed se incrementa el número de elementos procesados pero no en la misma proporción que la capacidad de procesamiento añadida. Las mismas limitaciones que se analizaron sobre la capa batch en 4.4.1.1 relativas al uso de una única máquina física y sobre la ejecución de una máquina virtual también son aplicables a estos resultados.

En la sección 4.4.2.2 se ha concluido que la escalabilidad de los dos primeros pasos es excelente, aunque ambos constituyen la parte que mayor tiempo consume de la ejecución de la capa speed.

4.5 Análisis global

En la sección 4.3 se ha analizado que, en este trabajo, se obtienen resultados de la calidad del clustering comparables o ligeramente superiores a los obtenidos en otros trabajos que realizan clustering sobre documentos aunque no trabajan en contextos Big Data. No ha sido posible comparar la calidad del clustering de documentos en contextos Big Data porque no se disponen de esas medidas en otros trabajos, ya que se centran en la escalabilidad.

En la sección 4.4 se ha analizado la escalabilidad de ambas capas, y se ha observado que aunque el sistema escala, el rendimiento, especialmente de la capa batch, es inferior al que se observa en otros experimentos que trabajan en contextos Big Data. La explicación de este peor rendimiento es que en los experimentos de este trabajo no se ha utilizado un cluster de máquinas sino una única máquina ejecutando una máquina virtual. Esto tiene un efecto negativo en el rendimiento especialmente al aumentar el número de cores, y por tanto se refleja negativamente en la escalabilidad.

En la (Tabla 4.7) y (Tabla 4.8) se muestran los valores de los tiempos de ejecución de cada capa, no tanto para un análisis cuantitativo, ya que los valores dependen de los recursos hardware sobre los que se ejecuta, sino para una comparación de los órdenes de magnitud que justifican si tiene sentido aplicar esta arquitectura. El tiempo necesario para ejecutar la fase batch sobre la colección de mayor tamaño se ha determinado experimentalmente que puede implicar varias horas (Figura 4.15). Si llega un nuevo documento nada más empezar la ejecución de esta capa, no se incorporará a los resultados de la aplicación en el peor de los casos hasta dos veces el tiempo de ejecución de la fase batch. En la capa speed el tiempo para procesar un documento está en el orden de centenas de milisegundos. A partir de estos valores se observa efectivamente la ventaja de obtener resultados del clustering en tiempo real con una calidad notable.

BATCH	
Tamaño de corpus (veces MovieLens 100k)	**Tiempo (minutos) para procesar todo el corpus**
256 veces	64
128 veces	20
64 veces	7
32 veces	3

Tabla 4.7. Resumen de tiempos de ejecución en Batch Layer (1 core)

SPEED	
cores	**Tiempo promedio (ms) para añadir un documento**
1	605
2	520
4	487

Tabla 4.8. Resumen de tiempos de ejecución de la ingesta en Speed Layer

El hecho de que se puedan incorporar nuevos documentos a los clusters y que se actualicen en tiempo real indica que la arquitectura lambda puede ser útil en aplicaciones donde sea necesaria una prontitud en los resultados.

Ejecutar ambas capas (junto a la capa proveedora) permite aprovecharse conjuntamente de las propiedades de cada una. La principal ventaja por utilizar una capa speed es la disminución del tiempo de latencia que permite obtener resultados en tiempo real. Adicionalmente a esta ventaja es una cualidad a destacar, especialmente en sistemas en producción, que al recalcularse todo a partir del conjunto de datos maestro es posible aplicar cambios de forma simple y recuperarse fácilmente frente a errores.

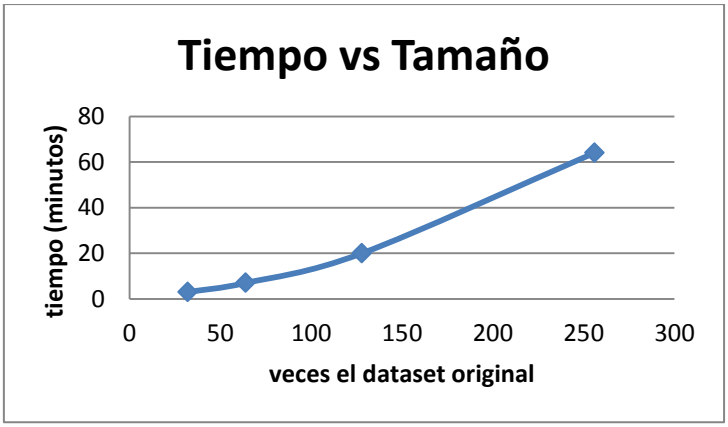

Figura 4.15. Tiempo de procesamiento frente al tamaño de la colección en Batch Layer (un core)

5 Conclusiones y trabajos futuros

En este trabajo se ha realizado un estudio de la arquitectura lambda aplicada al clustering de documentos sobre Big Data utilizando como tecnología Apache Spark. El clustering de documentos lleva implícito una problemática de elevada dimensionalidad y para sortearla se ha aplicado técnicas de reducción de dimensionalidad previas a la fase de clustering. Se han realizado experimentos para analizar la calidad del clustering y la escalabilidad de la solución propuesta.

La principal conclusión de este trabajo es que la utilización de la arquitectura lambda permite disponer de clustering de documentos sobre contextos Big Data con actualización de los resultados en tiempo real sin que se resienta la calidad, que es comparable a la que se obtiene en otros trabajos no orientados a Big Data pero que realizan funciones similares. No ha sido posible comparar la calidad del clustering de documentos frente a otros trabajos en contextos Big Data ya que estos se centran en la escalabilidad.

Esta arquitectura puede ser de utilidad en sistemas donde sea importante la inmediatez de los resultados como por ejemplo sitios web de noticias. Las noticias están vinculadas a temas del momento, que tienen unos términos cuyas relaciones evolucionan de forma dinámica con la actualidad, y este dinamismo se puede capturar y actualizar en tiempo real con el sistema descrito en esta propuesta.

Se ha observado experimentalmente que el tiempo de procesamiento de nuevos documentos puede estar en el orden del tiempo real, aunque está claro que depende del volumen de datos y de la infraestructura.

La escalabilidad de esta implementación es inferior a la que se obtiene en otros trabajos que realizan reducción de dimensionalidad y clustering sobre Big Data, lo que indica que todavía se puede afinar este aspecto ejecutando en un cluster de máquinas así como mejorar detalles de bajo nivel, con optimizaciones para incrementar el rendimiento.

Los experimentos de este trabajo se han ejecutado sobre una sola máquina ejecutando en paralelo gracias a sus múltiples cores. Esto ha provocado que los resultados de escalabilidad sean mejorables. Además el estudio de la escalabilidad se ha limitado al máximo de cores disponibles que eran 4. Esto tiene como consecuencia que las comparativas frente a otros trabajos también estén limitadas. Es un reto atrayente realizar el estudio sobre más CPUs en un cluster real, aunque requiere disponibilidad de recursos hardware.

El trabajo de (Hoffman et al., 2010) propone una versión online de LDA. Puede ser interesante en trabajos futuros estudiar su incorporación a la capa speed para evaluar de forma online la reducción de la dimensionalidad[13].

Finalmente, en futuras líneas de trabajo se puede estudiar la aplicación de la arquitectura lambda a clustering en BigData sobre otras áreas diferentes del clustering de documentos. También se puede aplicar a otras técnicas de la minería de datos (clasificación, recomendación, etc.) que trabajen sobre grandes volúmenes de datos dinámicos que requieran tiempos de respuesta cercanos al tiempo real.

[13] La última versión de Spark (1.5.0) ya dispone de la implementación de una versión online de LDA basado en (Hoffman et al., 2010).

Agradecimientos

Agradecimiento a mis directores de proyecto Raquel Martínez Unanue y Álvaro Rodrigo Yuste por sus valiosas observaciones, y su disponibilidad para ayudarme.

A mi familia, a mis amigos y compañeros de trabajo. A todos ellos por su paciencia y ánimos recibidos.

Bibliografía

Aggarwal, C. C., & Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series)*

Ailon, N., Jaiswal, R., & Monteleoni, C. (2009). Streaming k -means approximation. *Advances in Neural Information Processing Systems*, (pp. 10–18). http://papers.nips.cc/paper/3812-streaming-k-means-approximation.pdf

Achtert, E., Goldhofer, S., Kriegel, H. P., Schubert, E., & Zimek, A. (2012). Evaluation of clusterings--metrics and visual support. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference* on (pp. 1285-1288). IEEE.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics.

Asuncion, A., Welling, M., Smyth, P., & Teh, Y. W. (2009). On Smoothing and Inference for Topic Models, (Ml). *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (pp. 27-34). http://eprints.pascal-network.org/archive/00006729/

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463). New York: ACM press.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., & Vassilvitskii, S. (2012). Scalable K-Means ++. *Proceedings of the VLDB Endowment*, *5*(7), 622–633. http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

Banerjee, A., & Basu, S. (2007). Topic Models over Text Streams: A Study of Batch and Online Unsupervised Learning. In *SDM* (Vol. 7, pp. 437-442).

Beil, F., Ester, M., & Xu, X. (2002). Frequent term-based text clustering. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 436-442). ACM.

Bellman, R.E. (1961). *Adaptive control processes: a guided tour* (Vol. 4, p. 8). Princeton: Princeton university press.

Berkhin, P. (2002). Survey of Clustering Data Mining Techniques. *Technical Report, Accrue Software, San Jose, CA, 2002*.

Bíró, I., Siklósi, D., Szabó, J., & Benczúr, A. A. (2009). Linked latent dirichlet allocation in web spam filtering. In *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web* (pp. 37-40). ACM.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, *3*, 993–1022. doi:10.1162/jmlr.2003.3.4-5.993

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA* (pp. 205–218). USENIX Association. http://research.google.com/archive/bigtable-osdi06.pdf

Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*,275, 314-347.

Chen, Z., & Liu, B. (2014). Topic modeling using topics from many domains, lifelong learning and big data. In *Proceedings of the 31st International Conference on Machine Learning* (ICML-14) (pp. 703-711).

Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010). MapReduce Online. In *NSDI* (Vol. 10, No. 4, p. 20).

Córdova, P., & Spark, A. (n.d.). Analysis of Real Time Stream Processing Systems Considering Latency, 1–7.

Cortés, A., Téllez, A. E., Gallardo, M., & Peralta, J. J. (2014). Big Data Technology to Exploit Climate Information/Consumption Models and to Predict Future Behaviours. *International Technology Robotics Applications* (pp. 25-36). Springer International Publishing.

Da Silva Morais, T. (2015). Survey on Frameworks for Distributed Computing :, 95–105.

Dean, J., & Ghemawat, S. (2004). MapReduce: a flexible data processing tool. *OSDI, 2004*, (p. 1). http://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf

Dean, J., & Ghemawat, S.(2004) MapReduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004

Dhillon, I. S., & Modha, D. S. (2000). A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining* (pp. 245-260). Springer Berlin Heidelberg.

Diebold, F. X. (2003). 'Big Data'Dynamic factor models for macroeconomic measurement and forecasting. In *Advances in Economics and Econometrics: Theory and Applications, Eighth World Congress of the Econometric Society,"(edited by M. Dewatripont, LP Hansen and S. Turnovsky)(*pp. 115-122).

Dittrich, J., & Quian, J. (2012). Efficient Big Data Processing in Hadoop MapReduce. *Proceedings of the VLDB Endowment*, *5*, 2014–2015. doi:10.14778/2367502.2367562

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*, 78. doi:10.1145/2347736.2347755

Ene, A., Im, S., & Moseley, B. (2011). Fast Clustering using MapReduce. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 681–689. doi:10.1145/2020408.2020515

Engle, C., Lupher, A., Xin, R., Zaharia, M., Franklin, M. J., Shenker, Stoica, I. Berkeley, U. C. (2012). Shark: Fast Data Analysis Using Coarse-grained Distributed Memory. Sigmod, 1–4. doi:10.1145/2213836.2213934

Englert, S, J. Gray, T. Kocher, and P. Shah, (1989) A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scaleup on Large Databases *Tandem Computers, Technical Report* 89.4, Tandem Part No. 27469, May 1989

Fahad, a, Alshatri, N., Tari, Z., Alamri, a, Khalil, I., Zomaya, a, … Bouras, a. (2014). A Survey of Clustering Algorithms for Big Data: Taxonomy & Empirical Analysis. *IEEE Transactions on Emerging Topics in Computing*, 1–1. doi:10.1109/TETC.2014.2330519

Fan, W., & Bifet, A. (2013). Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, *14*(2), 1–5.

Fodor, I. (2002). A Survey of Dimension Reduction Techniques. *Technical Report UCRL-ID-148494*. doi:10.2172/15002155

Fresno Fernández, V.D. (2006). Representación Autocontenida de Documentos HTML: una propuesta basada en Combinaciones Heurísticas de Criterios (Doctoral dissertation, Universidad Rey Juan Carlos).

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1). Springer, Berlin: Springer series in statistics.

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: a review. *Sigmod Record*, *34*(2), (pp. 18–26).

Gama, J., Gaber M.M. (2007). Learning from Data Streams: Processing Techniques in *Sensor Networks* (p. 244).

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *ACM SIGOPS Operating Systems Review*, *37*(5), pp. 29–43. http://static.googleusercontent.com/media/research.google.com/es//archive/gfs-sosp2003.pdf

Girolami, M., & Kaban, A. (2003). On an Equivalence between PLSI and LDA Categories and Subject Descriptors. *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, 433–434. doi:10.1145/860435.860537

Gopalani, S., & Arora, R. (2015). Comparing Apache Spark and Map Reduce with Performance Analysis using K Means. *International Journal of Computer Applications*, *113*, 1–5.

Guha, S., Meyerson, A., Mishra, N., Motwani, R., & O'Callaghan, L. (2003). Clustering data streams: Theory and practice. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3), 515-528.

Havens, T. C., Bezdek, J. C., & Palaniswami, M. (2013). Scalable single linkage hierarchical clustering for big data. *Proceedings of the 2013 IEEE 8th International Conference on Intelligent Sensors, Sensor Networks and Information Processing: Sensing the Future, ISSNIP 2013*, *1*, 396–401. doi:10.1109/ISSNIP.2013.6529823

Henzinger, M. R., Raghavan, P., & Rajagopalan, S. (1998). Computing on data streams.

Hofmann, T. (1999). Probabilistic latent semantic indexing. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 50–57. doi:10.1145/312624.312649

Hofmann, T. (2001). Unsupervised Learning by Probabilistic Latent Semantic Analysis. *Machine Learning*, *42*(1-2), 177–196.

Hoffman, M. D., Blei, D. M., & Bach, F. (2010). Online Learning for Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems*, *23*, 1–9. doi:10.1145/1835804.1835928

Hotho, A., Staab, S., & Stumme, G. (2003). Wordnet improves Text Document Clustering. *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, *03*, 541–544. doi:10.1.1.8.8026

Jain, a. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, *31*(3), 264–323. doi:10.1145/331499.331504

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data* (Vol. 6). Englewood Cliffs: Prentice hall.

Jin, Y., Gao, Y., Shi, Y., Shang, L., Wang, R., & Yang, Y. (2011). P2lsa and p2lsa+: Two paralleled probabilistic latent semantic analysis algorithms based on the mapreduce model. In *Intelligent Data Engineering and Automated Learning-IDEAL* 2011 (pp. 385-393). Springer Berlin Heidelberg.

Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). *Learning Spark: Lightning-Fast Big Data Analysis*. " O'Reilly Media, Inc.".

Karau, H. (2013). *Fast Data Processing with Spark*. Packt Publishing Ltd.

Kitchin, R. (2014). *The data revolution: Big data, open data, data infrastructures and their consequences.* Sage.

Kumar, R., Gupta, N., Charu, S., & Jangir, S. K. (2014). Manage Big Data through NewSQL. *National Conference on Innovation in Wireless Communication and Networking Technology-2014*.

Labrinidis, A., & Jagadish, H. V. (2012). Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, *5*, 2032–2033. doi:10.14778/2367502.2367572

LANDAUER, Thomas K.; FOLTZ, Peter W.; LAHAM, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, *25*(2-3), p. 259–284.

Laney, D. (2001). 3D Data management: Controlling data volume, velocity and variety. *Application Delivery Strategies*, *949*(February 2001), 4. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

Larsen, B., & Aone, C. (1999). Fast and effective text mining using linear-time document clustering. *In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 16-22). ACM.

Lasserre, J., Bishop, C. M., & Minka, T. P. (2006). Principled hybrids of generative and discriminative models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (Vol. 1, pp. 87-94). IEEE.

Li, N., Zhuang, F., He, Q., & Shi, Z. (2012). Pplsa: Parallel probabilistic latent semantic analysis based on mapreduce. *Intelligent Information Processing VI*, 40–49. doi:10.1007/978-3-642-32891-6_8

Liu, X., Iftikhar, N., & Xie, X. (2014). Survey of real-time processing systems for big data. *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS '14*, 356–361. doi:10.1145/2628194.2628251

Loukides, M. (2010). What is Data Science? *O'Reilly Radar Report*, 1–10. http://cdn.oreilly.com/radar/2010/06/What_is_Data_Science.pdf

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval* (Vol. 1, p. 496). Cambridge: Cambridge university press.

Martínez-Prieto, M. A., Cuesta, C. E., Arias, M., & Fernández, J. D. (2014). The SOLID Architecture for Real-Time Management of Big Semantic Data. *Future Generation Computer Systems.*

Marz, Nathan; Warren, J. (2015). Big Data: Principles and best practices of scalable realtime data systems. *Manning Publications Co.*

Mayer-Schönberger, Victor; Cukier, K.; (2014). Big Data: A Revolution That Will Transform How We Live, Work, and Think. *International Journal of Advertising*. doi:10.1080/1369118X.2014.923482

Milios, E., Shafiei, M., Wang, S., & Zhang, R. (2000). A Systematic Study on Document Representation and Dimensionality Reduction for Text Clustering. *Cs.Usask.Ca*, 1–29. http://www.cs.usask.ca/~spiteri/DR_Proj_ver04.pdf

Moseley, B., & Marks, P. (2006). Out of the tar pit. *Software Practice Advancement (SPA)*, 1–66. doi:10.1.1.93.8928 http://shaffner.us/cs/papers/tarpit.pdf

O'Callaghan, L., Mishra, N., Meyerson, A., Guha, S., & Motwani, R. jeev. (2002). Streaming-Data Algorithms For High-Quality Clustering. *International Conference on Data Engineering*, (pp. 685–694).

Parsons L., Haque E., and Liu H. (2004) Subspace clustering for high dimensional data: A review. *SIGKDD Explorations, Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 2004.

Pentreath, N. (2015). *Machine Learning with Spark*.

Rajaraman, A., Ullman, J. D. (2012). *Mining of massive datasets* (Vol. 77). Cambridge: Cambridge University Press.

Reed, J. W., Yu, J., Potok, T. E., Klump, B. a., Elmore, M. T., & Hurson, A. R. (2006). TF-ICF: A new term weighting scheme for clustering dynamic data streams. *Proceedings - 5th International Conference on Machine Learning and Applications, ICMLA 2006*, 258–263. doi:10.1109/ICMLA.2006.50

Salton, G., Wong, a., & Yang, C. S. (1975). A Vector Space Model for Automatic Indexing. *Magazine Communications of the ACM*, *18*(11), 613–620. doi:10.1145/361219.361220

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5), 513-523.

Shafiei, M., Wang, S., Zhang, R., Milios, E., Tang, B., Tougas, J., & Spiteri, R. (2007, April). Document representation and dimension reduction for text clustering. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference* on (pp. 770-779). IEEE.

Shroff, G. (2013). *The Intelligent Web: Search, smart algorithms, and big data*. Oxford University Press.

Shih, C., Huang, C., Lin, B., Huang, C., Hu, W., & Cheng, C. (2014). Building a CDR analytics platform for real-time services. *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, 1–5.

Shirkhorshidi, A. S., Aghabozorgi, S., Wah, T. Y., & Herawan, T. (2014). Big Data Clustering: A Review. *Computational Science and Its Applications*, (ICCSA 2014), (pp. 707–720).

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *Mass Storage Systems and Technologies (MSST), IEEE 26th* , 1–10.

Singh, J. (2014). Big Data Analytic and Mining with Machine Learning Algorithm. *International Journal of Information and Computation Technology*, *4*(1), 33–40.

Smith M., Bill A., Boral H., Copeland G., Keller T., Schwetman H., Young C., (1989) An Experiment in Response Time Scalability, *Proceedings of the 6th International Workshop on Database Machines*, June, 1989

Soucy, P., & Mineau, G. W. (2005). Beyond TFIDF weighting for text categorization in the vector space model. In *IJCAI* (Vol. 5, pp. 1130-1135).

Steinbach M., Karypis G., and Kumar V. (2000) A comparison of document clustering techniques. In *KDD workshop on text mining* (Vol. 400, No. 1, pp. 525-526).

Strauch, C. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*.

Strehl, A., Ghosh, J., & Mooney, R. (2000). Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)* (pp. 58-64).

Tian, Y., Alagiannis, I., Liarou, E., Ailamaki, A., Michiardi, P., & Vukolic, M. (2014). DiNoDB : Efficient Large-Scale Raw Data Analytics Categories and Subject Descriptors. *InProceedings of the First International Workshop on Bringing the Value of Big Data to Users (Data4U 2014)*, (p. 1).

Vavilapalli VK et al. 2013. Apache Hadoop YARN: yet another resource negotiator

Wang, H., Wang, W., Yang, J., & Yu, P. S. (2002). Clustering by pattern similarity in large data sets. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* (pp. 394-405). ACM.

Wang, L., Sotiris, T., Teemu, R., & Kangasharju, J. (2015). Kvasir : Seamless Integration of Latent Semantic Analysis-Based Content Provision into Web Browsing. *Proceedings of the 24th International Conference on World Wide Web Companion*, (pp. 251–254).

Wang, Y., Bai, H., Stanton, M., Chen, W. Y., & Chang, E. Y. (2009). PLDA: Parallel latent dirichlet allocation for large-scale applications. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5564 LNCS*, 301–314. doi:10.1007/978-3-642-02158-9_26

Weinberger, K., Dasgupta, A., Attenberg, J., Langford, J., & Smola, A. (2009). Feature Hashing for Large Scale Multitask Learning. *Proceedings of the 26th Annual International Conference on Machine Learning*, (pp. 1113–1120). doi:10.1145/1553374.1553516

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1), 1-37.

Wu, X., Zhu, X., Wu, G.-Q., & Ding, W. (2014). Data Mining with Big Data. *Knowledge and Data Engineering, IEEE Transactions on*, *26*, 97–107. doi:10.1109/TKDE.2013.109

Xiao, Y. (2010). A Survey of Document Clustering Techniques Comparison of LDA and moVMF The Vector Space Model (VSM), 2–6.

Xu, R., & Wunsch II, D. (2005). Survey of Clustering Algorithms. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, *16*(3), 645–678. http://arxiv.org/abs/0912.2303

Xu, R., & Wunsch II, D. (2008). Clustering (Vol. 10). John Wiley & Sons.

Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, 10–10.

Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. (2011) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Technical Report UCB/EECS-2011-82, EECS Department, University of California*, Berkeley, 2011

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark : Cluster Computing with Working Sets. *HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 10. doi:10.1007/s00256-009-0861-0

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., ... & Stoica, I. (2012). Fast and interactive analytics over Hadoop data with Spark.*USENIX*; login, 37(4), 45-51.

Zhao, Y., Chen, Y., Liang, Z., Yuan, S., & Li, Y. (2014). Big Data Processing with Probabilistic Latent Semantic Analysis on MapReduce. *Training*, *1*, *3*.

Zhao, Weizhong; Ma, Huifang; He, Q. (2009). Parallel K -Means Clustering Based on MapReduce. *Cloud Computing. Springer Berlin Heidelberg*, *2009*, 674–679.

Zhong, S. (2005). Efficient streaming text clustering. *Neural Networks*, 18(5), 790-798.

Zikopoulos, P., & Eaton, C. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.

Anexo I

Hashing de rasgos

Para realizar la representación de los documentos se utiliza una forma rápida y eficiente en cuanto espacio de vectorizar los rasgos, esto es, de convertir rasgos arbitrarios en índices de un vector o matriz. Esta técnica se conoce como hashing de rasgos o *feature hashing*. Se basa en aplicar una función de hash a los rasgos y usar su valor como índice de una matriz dispersa en lugar de buscar los índices en un array asociativo. La técnica se describe en detalle en (Weinberger et al., 2009) utiliza una función de hash sobre los rasgos ($\phi: \chi \rightarrow \mathbb{R}^m$), lo cual permite por ejemplo almacenar un rasgo *w* (tal que $w \epsilon \mathbb{R}^d$) donde *d* es el tamaño del dominio del rasgo, requiere *O(d)* números, mientras que utilizando una función de hash podemos reducir el espacio a *O(m)*, donde *m* es el tamaño comprimido del dominio después de aplicar la función de hash.