Tecnologías de *big data*

PID_00247344

Jesús Alonso-Zárate
Francesc Julbe
Mario Macias Lloret
Jordi Nin

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya

Índice

Introducción

La disponibilidad de sensores, que provee al mundo de sentidos, para capturar datos del entorno y transmitirlos a través de internet, va a disparar la cantidad de datos disponibles.

Si hoy día ya tenemos muchos datos, lo que viene en el futuro es difícilmente imaginable.

Los datos, *per se*, no ofrecen ningún valor añadido; el valor se puede obtener de estos datos si se convierten en información que permite, finalmente, tomar decisiones inteligentes.

Principalmente, vamos a introducir el concepto de *big data* y las tecnologías disponibles para el *big data*. No es el objetivo de este material un análisis detallado de todas las partes de un sistema de *big data*, sino conocer la estructura general de un sistema de este tipo y adquirir un conocimiento básico de las funcionalidades que se requieren para tratar los datos y poder crear valor.

Objetivos

Los objetivos de este material son:

1) Conocer los elementos y la estructura de un sistema de *big data*.

2) Adquirir nociones básicas sobre procesamiento distribuido de datos.

3) Conocer las tecnologías existentes para el procesado de datos.

4) Conocer la existencia de las principales herramientas comerciales disponibles en el mercado.

5) Conocer las técnicas de análisis y visualización de datos en el contexto del *big data*.

1. Introducción a las tecnologías del *big data*

1.1. Definición de *big data*

Datos masivos, del inglés ***big data***, es el concepto con el que hacemos referencia a:

1) la identificación,
2) captura,
3) almacenamiento,
4) y procesamiento

de **grandes cantidades de datos**, con el objetivo último de **crear valor**.

A través de su procesamiento, los datos pueden pasar de ser simples datos a suponer una fuente de información.

Esta información puede devenir en conocimiento y este, eventualmente, en sabiduría. Finalmente, con el buen uso de la sabiduría, es posible generar valor.

Es importante, por lo tanto, tener en cuenta que los datos no tienen ningún valor en sí mismos; la búsqueda de **valor** a través de los datos debe ser la obsesión para cualquier aplicación del *big data* en el contexto de la industria 4.0.

El concepto **masivo** hace referencia a un volumen de datos suficientemente grande que supera la capacidad del software habitual para capturarlos, administrarlos y procesarlos en un tiempo razonable. El volumen a partir del cual los datos empiezan a considerarse masivos crece constantemente, y es difícil establecer una frontera entre lo que es *big data* y lo que no lo es. Por ejemplo, en el 2012 se estimaba su tamaño de entre una docena de terabytes hasta varios petabytes de datos en un único conjunto de datos.

En el contexto de la industria 4.0, no cabe duda de que los sensores y actuadores autónomos (cuyo uso se ha popularizado y masificado), que monitorizan, capturan y generan todo tipo de datos de manera continua, sí se pueden considerar como un sistema de *big data*.

Los escenarios propuestos por la mayoría de las aplicaciones relevantes para la industria 4.0 suponen uno de los mayores retos del *big data*: gestionar fuentes de información de diferente naturaleza.

Efectivamente, los datos se pueden clasificar según sean:

1) **Datos estructurados**: que suelen almacenarse en **bases de datos relacionales**. Un ejemplo de datos estructurados podría consistir en una base de datos que almacena los datos del DNI de personas en una tabla, con el número de campos de información y su formato predefinido.

2) **Datos no estructurados**: que suelen almacenarse en **bases de datos no relacionales** (las cuales describiremos con mayor detalle en el material «Bases de datos no relacionales»). Un ejemplo de datos no estructurados sería la documentación asociada a un perfil de un candidato para una plaza de trabajo, en el que se ha dado libertad para el formato del CV, así como para la información que se quiera adjuntar. La información agregada de muchos candidatos conforma un conjunto de datos no estructurados que deben ser tratados de manera conjunta.

3) **Datos semiestructurados**: un ejemplo de datos semiestructurados lo podemos ver en la información que se almacena en ficheros tipo XML o JSON. Los datos están organizados en una estructura siguiendo un formato de etiquetas predeterminadas, pero esta estructura puede variar de unos datos a otros, sin que garantice la consistencia entre todos los datos.

Las tecnologías de *big data*, por lo tanto, deben tener en cuenta esta diversidad del tipo de datos; este es uno de los grandes retos del *big data*.

El concepto de *big data*, muchas veces, se relaciona con las siguientes **cuatro V**:

1) **V de volumen**: gran volumen de datos, superior a los datos gestionados en aplicaciones «tradicionales». Este volumen de datos no es información y no sirve para generar valor. Estos datos deben ser procesados para poder extraer información, de la que crear valor.

2) **V de velocidad**: el tiempo de procesamiento de datos debe ser muy elevado para genera valor; en muchos casos, se deben tratar los datos en tiempo real para poder crear valor.

3) **V de variedad**: como hemos descrito antes, el *big data* tiene que gestionar muchos y variados tipos de fuentes de datos.

4) **V de veracidad**: dada la gran cantidad de datos, las técnicas de *big data* tienen que comprobar la veracidad de los datos recolectados; es necesario ejecutar sistemas que permitan validar la veracidad de los datos.

A partir de estas 4 V, cada aplicación podrá imponer unos u otros requisitos para el tratamiento de los datos.

1.2. El *big data* en la industria 4.0

Hoy día, podemos pensar en muchas aplicaciones en las que el *big data* juega un gran papel; por ejemplo:

1) **Redes sociales**: como es el caso de Facebook, donde el tratamiento de datos permite llevar a cabo publicidad muy personalizada para cada usuario.

2) **Gran consumo**: como por ejemplo Amazon, que personaliza sus campañas de marketing, o el caso de multitud de comercios que recogen información de sus clientes a través de tarjetas de fidelidad, con el fin último de conocer mejor sus gustos y patrones de compra, y poder así ajustar las ofertas y campañas de marketing de una manera más focalizada y, por lo tanto, eficiente.

3) **Telefonía**: los operadores móviles usan, cada vez más, datos de uso y de posición (localización) de sus clientes para ofrecer servicios de valor añadido o generar nuevas vías de beneficios.

4) **Política**: algunos políticos están empezando a usar información de preferencias de los votantes para poder diseñar sus campañas electorales y ganar el mayor número posible de votos.

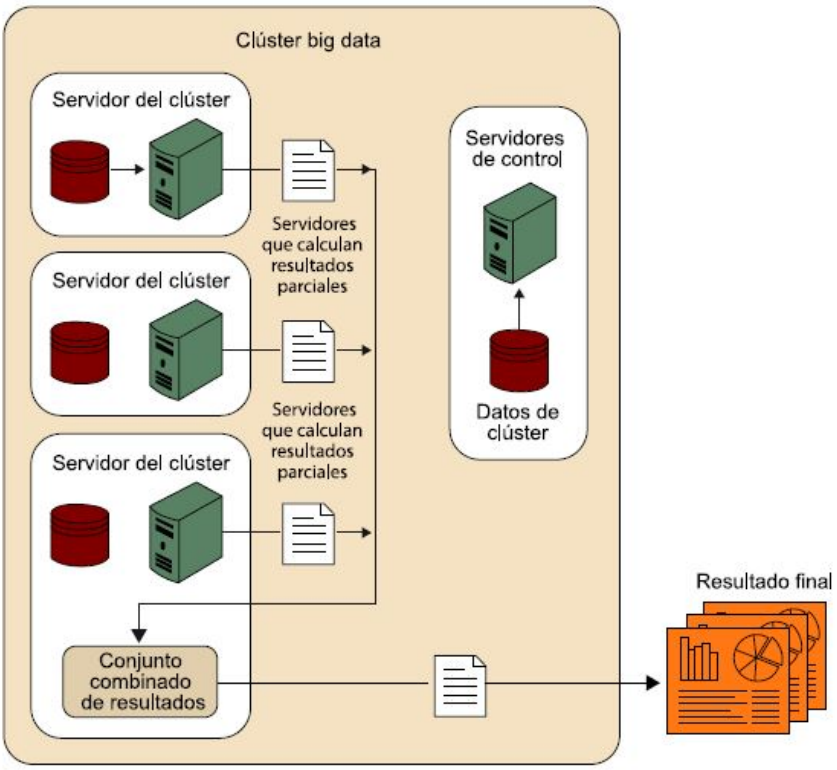
En el caso de la industria 4.0, no cabe duda de que el *big data* va a jugar un papel muy importante. La posibilidad de tratar datos recolectados para mejorar la eficacia y la eficiencia de procesos industriales, logística, transporte, etc. es muy grande. Las personas, los objetos, las cosas, los procesos; todo va a estar monitorizado y «datificado», y esto va a ser una fuente muy relevante de optimización que, finalmente, suponga la reducción de costes, nuevas vías de generación de ingresos, mayor seguridad, mayor fiabilidad, más satisfacción de las personas implicadas en los procesos productivos y mayor satisfacción del cliente.

1.3. Estructura general de un sistema de *big data*

Aunque las infraestructuras de *big data* tienen características específicas que adaptan el sistema a los problemas que deben resolver, todas comparten unos componentes comunes, como se describe en la figura 1.

Un sistema de *big data* consiste en un conjunto de servidores (ordenadores con capacidad de cómputo y procesamiento de datos) que forman un *cluster* de almacenamiento y computación de datos. También recibe el nombre de **centro de procesado de datos** (CPD).

Figura 1. Elementos de un sistema de *big data*



Este *cluster*, o conjunto de servidores, se gestiona a través de lo que se conoce como técnicas de computación distribuida. La **computación distribuida** es un modelo para resolver problemas de computación masiva, utilizando un gran número de ordenadores organizados en *clusters* e incrustados en una infraestructura de telecomunicaciones distribuida.

Cada servidor del *cluster* posee su propio disco, memoria RAM y CPU. Esto permite crear un sistema de cómputo distribuido con ordenadores heterogéneos y de propósito general, no diseñados de forma específica para crear *clusters*; lo cual reduce mucho los costes de estos sistemas de computación, tanto de creación como de mantenimiento.

Todos estos servidores se conectan a través de una red local. Esta red se utiliza para comunicar los resultados que cada servidor calcula con los datos que almacena localmente en su disco duro. La red de comunicaciones puede ser de diferentes tipos, desde Ethernet a fibra óptica, dependiendo de lo intensivo que sea el intercambio de datos entre los servidores.

Existen tres tipos de servidores:

1) Servidores que calculan **resultados parciales**. Estos servidores se ocupan de hacer los cálculos necesarios para obtener el resultado deseado en los datos que almacenan en su disco duro.

2) Servidores que combinan los diferentes resultados parciales para obtener el **resultado final** deseado. Estos servidores son los encargados de almacenar durante un tiempo los resultados finales.

3) Servidores de **control** o gestores de recursos, que aseguran que el uso del *cluster* sea correcto y que ninguna tarea sature los servidores. También se encargan de inspeccionar el hecho de que los servidores funcionen sin errores. En el caso de que detecten un mal funcionamiento, fuerzan el reinicio del servidor y avisan al administrador de que hay problemas en ciertos nodos.

El uso combinado de este conjunto de servidores es posible, como veremos más adelante, gracias al uso de un sistema de ficheros distribuido y la implementación de los algoritmos de procesamiento de datos en un entorno de programación distribuido, como por ejemplo **Hadoop** o **Spark**, que veremos más adelante.

1.4. Sistema de archivos distribuidos

El **sistema de archivos** o **sistema de ficheros** es el componente de un sistema de *big data* encargado de administrar y facilitar el uso del sistema de almacenamiento, ya sea basado en discos duros magnéticos (HDD, *hard disk drive*) o en memorias de estado sólido (SDD, *solid state disk*). En general, no es habitual usar almacenamiento terciario como DVD o CD-ROM para sistemas de *big data*.

Las funciones principales del sistema de archivos distribuidos son:

1) Asignar el espacio a los archivos.
2) Administrar el espacio libre.
3) Administrar el acceso a los datos almacenados.

Los sistemas de archivos estructuran la información almacenada en un dispositivo de almacenamiento de datos, y luego será representada, ya sea textual o gráficamente, utilizando un **gestor de archivos**.

La estructura lógica de los archivos suele representarse de forma jerárquica o en «árbol», utilizando una metáfora basada en la idea de carpetas y subcarpetas para organizar los archivos con algún tipo de orden. Para acceder a un archivo, se debe proporcionar su ruta (orden jerárquico de carpetas y subcarpetas) y su nombre de archivo seguido de una extensión (por ejemplo, .txt) que indica el contenido del archivo. Este tipo de ordenación lógica es la misma que tenemos en un ordenador personal al utilizar, por ejemplo, Microsoft Windows. El mismo concepto se usa para la ordenación de datos y archivos en un sistema de almacenamiento de archivos.

Cuando trabajamos con grandes volúmenes de datos, un único dispositivo físico de almacenamiento puede no ser suficiente. En estos casos, es necesario disponer de un sistema de archivos que permita almacenar información en múltiples dispositivos distribuidos en diferentes nodos (ordenadores), conectados entre sí mediante un sistema de red.

El problema de distribuir los datos es que puede suceder que los datos no estén almacenados en el mismo dispositivo (servidor, ordenador, etc.) donde se quieren llevar a cabo los cálculos. Por lo tanto, cada vez que tenemos que ejecutar un cálculo o procesamiento de datos, los datos deben ser copiados por la red de un nodo a otro. Este proceso es lento y costoso.

Para solucionar este problema, se creó el **Hadoop Distributed File System** (HDFS). Se trata de un sistema de archivos distribuido, escalable y portátil para el *framework* de cálculo distribuido Hadoop, creado especialmente para trabajar con ficheros de gran tamaño en entornos distribuidos.

Una de sus principales características es que usa un tamaño de bloque muy superior al habitual, para no perder tiempo en los accesos de lectura. Los ficheros que normalmente van a ser almacenados o ubicados en este tipo de sistema de ficheros siguen el patrón «write once, read many» ('escribe una vez y lee muchas'). Por lo tanto, está especialmente indicado para procesos *batch* de grandes ficheros, los cuales solo serán escritos una vez y, por el contrario, serán leídos gran cantidad de veces para poder analizar su contenido profundamente.

En HDFS se dispone de un sistema de archivos distribuido y especialmente optimizado para almacenar grandes cantidades de datos. De este modo, los ficheros son divididos en bloques de un mismo tamaño y distribuidos entre los nodos que forman el *cluster* de datos (los bloques de un mismo fichero se ubicarán en nodos distintos). Esto facilitará el cómputo en paralelo y evitará desplazar grandes volúmenes de datos entre diferentes nodos de una misma infraestructura de *big data*.

Esta división de la información entre diferentes nodos y su procesamiento distribuido da pie al concepto de sistemas de cálculo distribuido, que describimos en el siguiente apartado.

1.5. Sistema de procesamiento distribuido

Los sistemas de cálculo distribuido permiten la integración de los recursos de diferentes máquinas en red, y convierten la ubicación de un recurso en algo transparente al usuario.

El usuario accede a los recursos del sistema distribuido a través de un **gestor de recursos**, y se despreocupa de dónde se encuentra ese recurso y de cuándo lo podrá usar.

Aunque existen varias alternativas, hay dos sistemas de cálculo distribuido que están especialmente extendidos en el ecosistema del *big data*:

1) MapReduce.

2) Spark.

MapReduce es un modelo de programación introducido por Google en 1995 para dar apoyo a la computación paralela sobre grandes volúmenes de datos, con dos características principales:

1) Usa *clusters* de ordenadores.
2) Usa hardware no especializado.

El nombre de este sistema está inspirado en los nombres de sus dos métodos o funciones de programación principales: *map* y *reduce*.

MapReduce ha sido adoptado mundialmente, gracias a que existe una implementación *open source* denominada **Hadoop**, desarrollada por Yahoo, que permite usar este paradigma mediante el lenguaje de programación Java, de gran aceptación entre la comunidad de desarrolladores de software.

Desafortunadamente, no todos los análisis o el procesamiento de datos pueden ser calculados con este modelo de programación de MapReduce. Concretamente, solo son aptos aquellos que pueden calcularse como combinaciones de las operaciones de `map()` y de `reduce()`. Las funciones *map* y *reduce* están definidas respecto a datos estructurados en parejas del tipo (clave, valor).

En general, este sistema funciona muy bien para calcular agregaciones, filtros, procesos de manipulación de datos, estadísticas, etc.; operaciones todas ellas fáciles de paralelizar, que no requieren un procesamiento iterativo y en las que no es necesario compartir los datos entre todos los nodos del *cluster*.

MapReduce únicamente es capaz de distribuir el procesamiento a los servidores copiando los datos que hay que procesar a través de su disco duro. Esta limitación reduce el rendimiento de los cálculos iterativos, en los que los mismos datos tienen que usarse una y otra vez. Este tipo de procesamiento es básico en la mayoría de los algoritmos de aprendizaje automático.

Por su parte, el **proyecto de Spark** se centró desde el principio en mejorar las limitaciones de Hadoop. Spark mejora el comportamiento de las aplicaciones que hacen uso de MapReduce, y aumenta su rendimiento de manera considerable.

Spark es un sistema de cálculo distribuido para el procesamiento de grandes volúmenes de datos y que, gracias a su llamada «interactividad», hace que el paradigma

MapReduce ya no se limite a las fases `map()` y `reduce()`, y se puedan hacer más operaciones como *mappers*, *reducers*, *joins*, *groups by*, filtros, etc.

Spark proporciona una interfaz de programación de aplicación (API) para Java, Scala y Python, aunque es preferible que se programe en Scala, ya que es su lenguaje nativo.

La principal ventaja de Spark respecto a Hadoop es que guarda todas las operaciones sobre los datos en memoria. Esta es la clave de su buen rendimiento. Spark ofrece, por lo tanto, baja latencia computacional mediante el cacheo de datos en memoria y el hecho de que los algoritmos iterativos se ejecutan de forma eficiente compartiendo datos en memoria.

Tanto Hadoop como Spark están escritos en Java. Este factor común no es una simple casualidad, sino que tiene una explicación muy sencilla: los dos ecosistemas usan las **RMI** (*remote method invocation*) de Java para comunicarse de forma eficiente entre los diferentes nodos del *cluster*.

RMI es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java, y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java.

1.6. Gestión de recursos

Un **gestor de recursos distribuidos** es un sistema de gestión de colas de trabajos. Permite que varios usuarios, grupos y proyectos puedan trabajar juntos usando una infraestructura compartida como, por ejemplo, un *cluster* de computación.

Apache Mesos, por ejemplo, es un gestor de recursos que simplifica la complejidad de la ejecución de aplicaciones en un conjunto compartido de servidores. En su origen, Mesos fue construido como un sistema global de gestión de recursos, completamente agnóstico sobre los programas y servicios que se ejecutan en el *cluster*. Bajo esta idea, Mesos ofrece una capa de abstracción entre los servidores y los recursos. Es decir, lo que ofrece Mesos es, básicamente, un lugar donde ejecutar aplicaciones sin preocuparnos de los servidores que tenemos por debajo.

Otro gestor de recursos es **YARN**.

YARN (*Yet Another Resource Negotiator*) nace para dar solución a una idea fundamental: dividir las dos funciones principales del JobTracker. Es decir, tener la gestión de recursos en servicios totalmente separados e independientes, por un lado y, por otro, la planificación y monitorización de las tareas o ejecuciones.

Un algoritmo de MapReduce, por sí solo, no es suficiente para la mayoría de los análisis que Hadoop puede llegar a resolver. Con YARN, Hadoop dispone de un entorno de

gestión de recursos y aplicaciones distribuidas en el que se pueden implementar múltiples aplicaciones de procesamiento de datos totalmente personalizadas y específicas, para llevar a cabo una gran cantidad de análisis de forma concurrente.

2. Tipología de procesamiento distribuido de datos en *big data*

2.1. La computación distribuida

Entendemos por **procesado distribuido** (en inglés, *distributed computing*) la capacidad de compartir recursos (recursos de computación, almacenamiento, memoria) entre una red de ordenadores para la realización de una serie de operaciones y/o tareas.

Esta arquitectura es muy flexible, ya que pueden añadirse o quitarse computadoras a la red (comúnmente denominadas nodos) sin que esto tenga un impacto significativo en su funcionamiento. Dichos nodos pueden estar cerca o lejos geográficamente, solo es necesario que estén conectados entre sí (mediante una conexión IP de internet, por ejemplo).

El procesado distribuido de datos puede contribuir a la reducción de costes, ya que la carga de computación se distribuye entre los diferentes equipos de la red, en contraste con la adquisición de grandes equipos dedicados y centralizados en un CPD (centro de procesado de datos). Hay que tener en cuenta que la computación distribuida permite evitar los costes de mantenimiento y actualización de equipos dedicados, y optar por el pago por uso según las necesidades reales de computación. De algún modo, es como el concepto de *shared economy* aplicado al mundo de la computación y la capacidad de procesado de datos.

Además, si la red está adecuadamente configurada, el procesamiento de datos distribuido es fiable, ya que la carga de computación puede balancearse (repartirse) entre nodos.

Además, los datos estarán almacenados y replicados entre nodos, lo que permite tolerancia a fallos de algunos nodos (también configurable) sin comprometer el resultado de una tarea concreta. En aplicaciones en las que la fiabilidad es necesaria, la alternativa distribuida resulta de gran interés.

Encontramos diferentes maneras de llevar a cabo procesado distribuido de datos, que presentamos en los apartados siguientes.

2.2. Procesado de datos en modo *batch*

El **procesado de datos en modo *batch*** es un modo de procesar grandes volúmenes de datos, en el que una serie de procesos (lo que puede llamarse transacción) se ejecutan sobre un conjunto de datos.

Durante la ejecución de un proceso en *batch*, el usuario rara vez tiene que interactuar con el proceso. La ejecución requerirá unos *inputs*, y típicamente producirá un *output* una vez que haya finalizado el proceso. Su duración puede ser muy variable, desde minutos hasta horas.

El modo de procesado *batch* es, hoy día, el más ampliamente utilizado. Efectivamente, son pocos los procesos empresariales de criticidad suficiente que requieran procesado de datos en tiempo real. Sin embargo, en el caso de la industria 4.0, esto va a cambiar, ya que en muchas aplicaciones será necesario llevar a cabo un procesado de datos en tiempo real. En este caso, usaremos técnicas de procesado en *streaming*, como veremos en apartados posteriores.

Un proceso en modo *batch* se caracteriza por lo siguiente:

- Tiene acceso a todos los datos o a una gran cantidad de ellos.
- Suele llevar a cabo operaciones grandes y complejas.
- Es un proceso más enfocado al *throughput* (cantidad de datos que se pueden tratar) que al propio tiempo de ejecución (o latencia).
- Su latencia suele medirse en minutos u horas.

Si en el problema de cálculo al que nos enfrentamos el volumen de datos es el factor clave –el KPI (*key performance indicator*)–, el modo de procesado adecuado es el modo *batch*.

Para resumir, el procesado de datos en *batch* es muy eficiente en el tratamiento de grandes volúmenes de datos. Los datos se introducen en el sistema y se procesan, de modo que generan unos *outputs*. En este tipo de procesado, el tiempo invertido no debe suponer un problema prioritario. Los trabajos están configurados para ejecutarse sin intervención manual y, dependiendo del tamaño de los datos que se procesan y la potencia de cálculo del sistema, la salida se puede retrasar de manera significativa. Se trata, por lo tanto, de un método de análisis fuera de línea.

2.3. Procesado de datos en modo tiempo real: *streaming*

Aunque hoy día muchos de los escenarios de procesado de datos utilizan el modo *batch*, en algunas ocasiones aparece la necesidad de procesar datos en *streaming* o (casi) en tiempo real (*real-time*). En el caso de la industria 4.0, este tipo de aplicaciones jugará un papel fundamental.

Por ejemplo, en el caso de un sistema de detección de fallos en una planta de fabricación en línea, debe detectarse en tiempo real para o bien detener la línea, o bien aplicar las acciones correctivas lo antes posible, con el fin de minimizar los posibles daños o pérdidas de eficiencia en el proceso de fabricación.

Las deficiencias y los inconvenientes de procesamiento de datos en *batch* fueron ampliamente identificados por la comunidad de *big data*, y se hizo evidente que el procesamiento en casi tiempo real o *streaming* es una necesidad en muchas aplicaciones prácticas.

En los últimos años, algunas soluciones han aparecido para cubrir esta necesidad, como Apache Storm de Twitter, Spark Streaming, Apache Flink o S4 de Yahoo.

Es importante tener en cuenta que aunque se mencione el concepto «tiempo real», se produce un abuso del lenguaje ya que, en el mundo de la computación, tiempo real se asocia a procesos y eventos causa-efecto que se mueven en el orden de los milisegundos o microsegundos, mientras que en los escenarios de *big data*, la respuesta se moverá en el orden de los segundos o minutos, como mucho. Este es el motivo por el que se utiliza el término «casi» cuando hablamos de tiempo real en el procesado de datos en *streaming*.

Podríamos caracterizar el procesado en *streaming* de la siguiente forma:

- Lleva a cabo un cálculo o proceso sobre una porción de los datos relativamente pequeña y seleccionada.
- El cálculo es relativamente simple. En caso contrario, perdería velocidad.
- Necesita completar cada cálculo en «casi» tiempo real, probablemente del orden de pocos segundos.
- Los cálculos son generalmente independientes, y no forman parte de una cadena de procesado compleja.
- Es asíncrono, es decir, la fuente de datos no interactúa con el proceso y no hay señales de sincronización entre ellos.

En resumen, el procesado de datos en *streaming* se focaliza en la velocidad de lectura y el análisis de los datos, en contraposición con la gran capacidad de procesado de

datos de los métodos en *batch*. En el caso *streaming*, los datos deben ser procesados dentro del periodo de tiempo pequeño o casi en tiempo real. El procesado en *streaming* permite la toma de decisiones rápidas basadas en tendencias que se desarrollan en ventanas de tiempo pequeñas.

2.4. Procesado de datos según eventos complejos: CEP

El procesamiento según eventos complejos (o *complex event processing*, CEP) puede entenderse como un subconjunto del procesado de datos en *streaming*.

El **CEP** consiste en la capacidad de predecir eventos de alto nivel que pueden derivarse de un conjunto específico de factores de bajo nivel. CEP identifica y analiza las relaciones de causa y efecto entre los eventos en casi tiempo real, de modo que permite la toma de decisiones y medidas eficaces en respuesta a situaciones específicas.

CEP se utiliza en la gestión de la política de seguridad de riesgos, la gestión de relaciones con clientes (en los denominados CRM), servidores de aplicaciones y, especialmente, en situaciones que implican numerosos factores que interactúan de forma variable, tales como la inversión y los entornos de préstamo para las instituciones financieras, o en la gestión de amenazas para redes de comunicaciones.

La idea que subyace en una aplicación CEP es el análisis de diferentes eventos para encontrar patrones entre ellos. Aunque los términos pueden confundirse, podría decirse que un sistema CEP se construye generalmente encima de un sistema de procesado en *streaming*. Las consultas de los sistemas *streaming* son de más bajo nivel y pueden operar en ventanas de tiempo corto, mientras que las consultas de los sistemas CEP pueden ser más complejas y tomar periodos de tiempo más largo.

Para resumir, el procesamiento de eventos complejos se ocupa de eventos discretos y puede entenderse como un caso concreto dentro del procesado de datos en *streaming*.

3. Tecnologías específicas de procesado de datos

3.1. Tecnologías de procesado en modo *batch*

Una de las tecnologías pioneras y más destacadas de procesado en modo *batch* es el entorno de Hadoop (Hadoop *framework*).

El *framework* Hadoop está formado por un conjunto de servicios destinados a ofrecer funcionalidades de procesado distribuido sobre un *cluster* formado por nodos que comparten sus recursos de computación (CPU), memoria y almacenamiento.

En el 2003, Google se enfrentó al desafío de indexar toda la World Wide Web (WWW). Por un lado, la Web crecía a un ritmo muy elevado y, por otro lado, ya habían alcanzado el límite de escalabilidad de sus soluciones de datos relacionales (*relational database management system*, RDBM). De este modo, Google buscó y encontró un nuevo enfoque para poder procesar y analizar grandes cantidades de datos. Este enfoque se basaba en dos pilares:

1) **GFS (*Google file system*)**: es un sistema de ficheros distribuido entre todos los nodos de un *cluster*. Este sistema aportaba dos funcionalidades: primero, los datos se analizan localmente, allí donde están almacenados; y segundo, los datos están replicados a través de los nodos del *cluster*, lo que da tolerancia a fallos y, además, abstrae la complejidad de la infraestructura para el desarrollador con un punto de entrada único al sistema de ficheros, aunque estos estén distribuidos.

2) **MapReduce**: es un paradigma de computación distribuida en el que una tarea se descompone en pequeñas subtareas, independientes entre sí, que se ejecutan en los nodos del *cluster*, para ser agregadas en un proceso posterior, una vez hayan finalizado todas las subtareas.

Esta arquitectura estaba enfocada a sacar partido de las arquitecturas más predominantes en la época: cantidades de memoria RAM limitada y grandes capacidades de disco duro. En esta arquitectura, el programador solo debe ocuparse de la lógica del procesado, mientras que el propio *framework* de Hadoop lleva a cabo las funciones de gestión de la infraestructura que soporta dicha lógica.

Apache Hadoop fue la primera iteración *open source* del nuevo enfoque introducido por Google, y se centró en tres grandes prioridades:

1) **Fiabilidad y tolerancia a fallos**: existe replicación de datos entre nodos, y la gestión de réplicas es transparente al desarrollador.

2) **Económica**: el *framework* no requiere grandes inversiones en costosos equipos, ya que puede ejecutarse en hardware estándar, denominado *commodity hardware*.

3) **Escalabilidad**: el *cluster* permite añadir nuevos nodos de forma sencilla, y puede llegar a miles de nodos; de nuevo, es transparente para el programador.

El *framework* Hadoop ofrece servicios clave sobre los que operan otros servicios y herramientas *big data*.

A partir del paradigma MapReduce y del *framework* Hadoop, muchas herramientas han aparecido para cubrir distintas necesidades. De entre todas las plataformas y herramientas *big data* aparecidas tras la irrupción de Hadoop, la más destacable es Apache Spark.

Apache Spark se presenta con el objetivo de solucionar tres problemas clave:

1) **Usabilidad**: escribir aplicaciones en Spark es realmente sencillo, y la configuración de un *cluster* Spark es relativamente fácil, y mejora significativamente el desarrollo de aplicaciones *big data* más allá de MapReduce.

2) **Rendimiento**: Spark resuelve uno de los grandes problemas de MapReduce: su rendimiento. Sin duda, a pesar de lo novedoso del paradigma MapReduce, su rendimiento es bajo, debido a que trabaja esencialmente a partir de ficheros, además de estar sujeto a un modelo de difícil implementación. Sin embargo, Spark hace uso de la memoria RAM del sistema, y saca provecho de las arquitecturas modernas, en las cuales la memoria RAM ya no es un recurso tan costoso como lo fue anteriormente.

3) **Simplicidad**: Spark se presenta como un motor de procesado distribuido, más allá de una herramienta específica. Esto permite ejecutar múltiples herramientas «sobre» él. Hasta ahora, algunas funcionalidades de consulta o procesado requieren el uso de diferentes herramientas. Sin embargo, Spark ofrece muchas funcionalidades agrupadas en un conjunto de herramientas, como SparkSQL, *streaming*, *machine learning* o GraphX. Antes de Spark, una operación que pudiera requerir extraer datos de una fuente de datos diversa, aplicar algún algoritmo de *machine learning* y finalmente hacer consultas sobre esos datos podía requerir tres motores distintos: Sqoop o Flume, Mahout para las tareas de *machine learning*, y Hive o Impala como herramientas de consultas sobre entornos distribuidos. Spark, sin embargo, ofrece esta funcionalidad a través de las distintas herramientas del *framework*, con total interactividad entre ellas.

Spark extiende el paradigma *MapReduce* y ofrece más modelos de computación, facilita la realización de tareas más complejas e incrementa la velocidad de procesado hasta dos órdenes de magnitud, gracias a un uso de memoria mucho más intenso que Hadoop.

Por otro lado, Spark hace uso del modelo de computación DAG (*directed acyclic graph*); se trata de un modelo de ejecución de tareas en el que estas son vértices de un grafo, y el orden de ejecución se especifica por la direccionalidad de los ejes

del grafo. Acyclic significa que no hay bucles en el grafo. Además, en un modelo de computación DAG, los procesos se ejecutan en paralelo. Este modelo ofrece una mayor flexibilidad para implementar computaciones en modo *pipeline*, mucho más avanzado que el modelo MapReduce.

3.2. Tecnologías de procesado en modo *streaming*

Podemos destacar cuatro tecnologías clave de procesado de datos en modo *streaming*:

1) Apache Flume.
2) Apache Kafka.
3) Spark Streaming.
4) Apache Storm.

Las describimos brevemente a continuación.

3.2.1. Apache Flume

Apache Flume es una herramienta cuyas principales funcionalidades son reunir, agregar y mover grandes volúmenes de datos provenientes de distintas fuentes hacia el repositorio HDFS en casi tiempo real.

Se trata de una herramienta útil en situaciones en las que los datos se crean de forma regular y/o espontánea. A medida que nuevos datos aparecen, ya se trate de *logs* o datos de otras fuentes, estos se almacenan en el sistema distribuido de manera automática.

El uso de Apache Flume no se limita a la agregación de datos desde *logs*. Debido a que las fuentes de datos son configurables, Flume permite ser usado para recoger datos desde eventos vinculados al tráfico de red, redes sociales, mensajes de correo electrónico o casi cualquier tipo de fuente de generación de datos dinámica.

3.2.2. Apache Kafka

Apache Kafka es un proyecto Apache originariamente desarrollado por Linkedin. Es un sistema distribuido de publicación-suscripción de mensajes, y está diseñado para procesar en tiempo real la actividad en un *stream* de datos.

Desde un punto de vista terminológico, en Kafka existen los:

1) ***Topics***: categorización de la información por Kafka. Un *topic* es una categoría en la cual se publican mensajes.

2) ***Producers***: procesos que publican mensajes acerca de un *topic*.

3) ***Consumers***: procesos consumidores de la información acerca de un *topic*, a partir de una suscripción a esta categoría.

4) ***Broker***: el *cluster* Kafka, consistente en uno o más servidores.

El funcionamiento es el siguiente: para cada *topic*, el *broker Kafka* mantiene la información particionada, distribuida y replicada entre los servidores del *cluster*. Cada partición es una secuencia inmutable y ordenada de mensajes que se actualiza de forma continua, y cada mensaje en la partición está identificado con un *id*, que lo identifica de forma unívoca. Todos los mensajes se guardan en el servidor Kafka durante un periodo de tiempo, de manera independiente de si se han consumido o no. El particionamiento de la información permite paralelismo y escalabilidad.

Cada servidor *Kafka* cuenta con un proceso líder, que se encarga de gestionar todas las lecturas y escrituras de mensajes. Otros procesos denominados *followers* imitan la actividad del líder y permiten que, en caso de fallo del líder, un *follower* pueda reemplazarle y mantener el estado del sistema actualizado. Además, para cada servidor, las particiones tienen líderes y *followers* distintos, de modo que se balancea la carga en el *cluster*. Los *producers* o productores se encargan de publicar mensajes en cada *topic*, y pueden elegir en qué partición publicar el mensaje. Finalmente, cada mensaje publicado en un *topic* es entregado a un solo consumidor «suscrito» a ese *topic* de información.

El paralelismo en Kafka viene determinado por el particionamiento de la información. Si un *topic* tiene N particiones, el grupo de consumidores (procesos que procesan la información) puede asignar un máximo de $N_{threads}$ de procesado en paralelo. Así, si un *topic* está formado por 10 particiones y tenemos 15 consumidores, solo 10 consumidores van a leer información de este *topic*, mientras que 5 permanecerán inactivos.

3.2.3. Spark Streaming

Spark Streaming es una API de Spark cuyo objetivo es el procesado de un flujo continuo de datos que son consumidos por la aplicación. En esta API, la capa de abstracción para trabajar con el flujo continuo de datos la ofrece el llamado DStream *(discretized stream*), que es una secuencia de RDD (*resilient distributed dataset*) que representa el flujo continuo de datos como pequeños procesos *batch*. De este modo, el DStream en Spark Streaming toma el papel del RDD en el núcleo de Spark.

3.2.4. Apache Storm

Apache Storm es un *framework* competidor de Spark Streaming. Storm ofrece una latencia muy baja (menor que la de Spark), con capacidad de análisis distribuido de

datos en tiempo real y *machine learning*. Además, es una solución altamente escalable y tolerante a fallos.

Apache Storm ofrece buenas funcionalidades en cuanto a fiabilidad del sistema, y garantiza que cada paquete de datos del *stream* se leerá como mínimo una vez. Además, garantiza que un paquete de datos se leerá solo una vez, funcionalidad que también ofrece Spark Streaming. Sin embargo, Spark no puede garantizar que en caso de fallo de un nodo no se vayan a perder datos que este nodo tenga almacenados en memoria en ese momento. Pueden almacenarse datos en HDFS de forma temporal para garantizar resistencia a fallos, pero penaliza el rendimiento del sistema. Storm garantiza la lectura de todos los *streams* de entrada. Sin embargo, la fiabilidad del sistema dependerá de la fiabilidad de la fuente continua de datos. Como ejemplo, Kafka es fiable, mientras que Twitter no es fiable.

Storm es compatible con muchos lenguajes de programación como Java, Scala, Python, Ruby o Clojure, entre otros. Sin embargo, Spark provee una mayor integración con todo el resto del *framework* Spark. Hasta el momento, Storm tiene una base de usuarios importante y ha dejado de ser un proyecto en incubación. Tanto Hortonworks como MapR lo integran en su distribución, pero Cloudera no lo hace. Por tanto, es difícil saber cuál será el framework predominante para procesar datos en *streaming* a medio y largo plazo. A pesar de la comunidad de usuarios de Storm, es una herramienta especializada más dentro del ecosistema *big data* y Hadoop, mientras que Spark Streaming cuenta con el apoyo de todo el *framework* Spark y está plenamente integrado en el mismo.

3.3. Otras herramientas

A continuación, vamos a presentar brevemente algunas herramientas que merecen una especial atención en el entorno *big data*, ya sea por su peso histórico (Mahout, principalmente) o por ser herramientas en desarrollo pero que introducen nuevas funcionalidades que potencialmente pueden tener mucha aceptación en el mundo *big data*.

3.3.1. Mahout

Mahout es un proyecto de Apache, cuyo objetivo es ofrecer capacidad de llevar a cabo operaciones de *machine learning* en entornos distribuidos sobre Hadoop. De este modo, las operaciones que se deben llevar a cabo se dividen en tareas MapReduce que, en muchas ocasiones, pueden ser una secuencia de las mismas.

Hasta la aparición de Spark, Mahout era una de las mejores plataformas para este tipo de computación. Sin embargo, debido a la complejidad de las tareas que se llevan a cabo y la rigidez del modelo MapReduce, el tiempo de cálculo era muy grande y la necesidad de secuenciar cadenas de operaciones MapReduce generaba una gran cantidad

de datos intermedios (cabe recordar que MapReduce es un modelo de computación en el que el intercambio de datos se basa en ficheros).

Spark mejoró en órdenes de magnitud los tiempos de cálculo para hacer la misma operación, y dejó a Mahout en una situación que ha obligado a replantearse su arquitectura.

Actualmente Mahout ya trabaja, pese a tratarse de versiones en desarrollo, sobre Spark, a través del nuevo entorno Samsara.

3.3.2. BlinkDB

BlinkDB es una herramienta todavía en desarrollo, pero que introduce una interesante novedad: permite hacer consultas similares a SQL (casi idénticas a las utilizadas en Hive y SparkSQL) sobre entornos distribuidos, aceptando márgenes de error. En otras palabras, es posible hacer una consulta en la que se acepta que los resultados devueltos tengan un cierto margen de error. **El beneficio principal es que reduce de forma drástica los tiempos de computación**.

Para situaciones en las que se está haciendo exploración de datos distribuidos, el científico de datos puede tolerar cierto margen de error si, a cambio, se reduce el tiempo de consulta en varios órdenes de magnitud.

3.3.3. Apache Flink

Apache Flink es otra plataforma para procesar datos en *streaming* de baja latencia, similar a Spark Streaming o Storm.

Sus funcionalidades son muy parecidas a Spark Streaming. Es capaz de procesar datos en *streaming* o fuentes estáticas como ficheros de un sistema HDFS. Tiene sus propias API para trabajar sobre colecciones (pueden hacerse transformaciones, agrupaciones, filtrado, etc.).

También tiene librerías de *machine learning*, API para trabajar con grafos y API para trabajar con datos estructurados con funcionalidades de consulta SQL.

Comparándolo con las dos herramientas presentadas anteriormente para *streaming* (Spark y Storm), Flink se sitúa entre las dos aproximaciones:

- Ofrece capacidades de procesar datos en *batch*, algo que no es posible en Apache Storm.

- Es una plataforma que trabaja en *streaming* puro, no como Spark, que trabaja en *minibatches*.

3.3.4. Elasticsearch

Elasticsearch es una herramienta que permite indexar un gran volumen de datos y, posteriormente, hacer consultas variadas como búsquedas aproximadas, búsquedas de texto completo, texto resaltado, etc. Se basa en Lucene, un motor de búsqueda de texto escrito en Java.

La velocidad de las búsquedas es muy rápida, ya que los datos están indexados. Su funcionalidad se ofrece a través de una interfaz REST, intercambiando datos con formato JSON y ocultando el núcleo de Lucene. La interfaz REST es accesible con cualquier lenguaje como Java, Python o PHP, entre otros.

Elasticsearch puede trabajar con Hadoop y Spark.

4. Ejemplos de plataformas de datos comerciales

A continuación, listamos cinco soluciones propuestas por algunos de los principales proveedores de soluciones hardware y software del mercado, como son:

1) Amazon.
2) IBM.
3) Google.
4) Microsoft.
5) Hewlett-Packard.

El modelo propuesto por todos ellos es parecido: uso de *clusters* desplegables bajo demanda en la nube, y basados en tecnologías Hadoop y Spark.

4.1. Amazon

Amazon es una de las empresas que más fuerte han apostado por las plataformas de procesado distribuido, y ha integrado un paquete de servicios de procesado en su plataforma Amazon WS (Web Services).

Concretamente, se incluye Amazon EMR (Amazon Elastic MapReduce): una plataforma que permite la creación dinámica de *clusters* Hadoop, incluyendo todas las herramientas ya presentadas para el procesado *big data*, como Hive, Impala, Spark, etc.

La capacidad de cálculo la proporciona el servicio EC2 (*elastic compute cloud*), que permite la creación de *clusters* virtuales llamados instancias, y que está totalmente personalizado a las necesidades del usuario (plataformas Linux o Windows, y recursos de memoria y computación). La capacidad de almacenamiento la provee el servicio S3 (*simple storage service*).

De este modo, la solución completa de procesado podría estar formada por el sistema de almacenamiento S3, mientras que EMR (formado por instancias EC2) lee y escribe datos en el sistema S3.

Amazon cobra por el consumo de los recursos; cuando una tarea de procesado finaliza, el *cluster* se apaga.

4.2. IBM Analytics

IBM también ofrece soluciones para procesado de *big data*, a través de la plataforma IBM Analytics.

Se trata de un paquete ofimático de herramientas para tratamiento de datos *big data*, como:

1) IBM SPSS Modeler, para desarrollar modelos predictivos.

2) IBM Cognos Business Intelligence, para funcionalidades de *business intelligence*.

3) Herramientas de gestión de contenidos (ECM, *enteprise content management*).

4) Almacenamiento de datos en la nube (similar a S3 de Amazon).

5) *Data warehousing*.

6) IBM Open Platform; servicios de despliegue de *clusters* Hadoop de forma dinámica, entre los cuales incluye varias de las aplicaciones ya descritas en este material, como Spark, Kafka, Flume, Pig, Hive, etc.

En el núcleo de su plataforma, IBM ha hecho una apuesta por Apache Spark como motor de procesado de datos distribuidos.

4.3. Google

Google también ofrece servicios de nube para el despliegue de *clusters* Hadoop y Spark.

Cloud DataProc, solución integrada en Google Cloud Platform, permite desplegar *clusters* Hadoop y Spark, incluyendo las herramientas del ecosistema ya estudiadas (Pig, Hive, etc.).

Como en todos los productos y herramientas de Google, su solución está muy enfocada a la **facilidad de uso**.

Además, ha implementado servicios adicionales para consultas SQL como Google BigQuery, similar en funcionalidades a SparkSQL, o herramientas para la definición de *pipelines* como Cloud Dataflow.

Su plataforma incluye un paquete de *machine learning*, todavía en fase de desarrollo, con una tecnología propia llamada TensorFlow. TensorFlow propone un modelo de procesado distribuido basado en un grafo en el que los vértices son operaciones, y los ejes son tensores (vectores multidimensionales). El grafo define el *dataflow* completo, y puede ejecutarse en CPU o en GPU.

GPU

GPU, o unidad de procesamiento gráfico (*graphics processor unit*), es un coprocesador dedicado al procesamiento de gráficos.

Como en el caso de Amazon, Google factura por uso.

4.4. Microsoft Azure y HDInsight

Dentro del paquete de servicios Microsoft Azure, que es una solución para ofrecer servicios en un ámbito empresarial en la nube, Microsoft incluye su paquete Microsoft HDInsight.

HDInsight es una distribución de Hadoop en la nube que ofrece servicios de procesado y que integra la mayoría de las herramientas presentadas en este material, como por ejemplo operaciones *batch* con MapReduce, SDQL con Hive, NoSQL con HBase, Spark, Storm, etc. En este caso, se utiliza la distribución de Hortonworks de Hadoop.

El sistema ofrece capacidad de escalado en la nube, y crea *clusters* Hadoop bajo demanda.

Su uso se tarifica mensualmente, dependiendo de la configuración deseada y de los servicios que hay que utilizar en el *cluster*.

4.5. HPE Vertica

Hewlett-Packard Enterprise (HPE) también comercializa soluciones *big data*.

Como fabricante de hardware, HPE ofrece soluciones de infraestructura especialmente diseñadas para desplegar *clusters* Hadoop, y soporta las distribuciones de Hortonworks y Cloudera, con especificaciones de servidores HPE.

Desde el punto de vista de software de análisis de datos en entornos distribuidos, comercializa la solución Vertica, la cual se basa en un motor de consulta SQL orientado a columna y que permite llevar a cabo consultas sobre *clusters* Hadoop.

La solución Vertica OnDemand ofrece capacidad de análisis de *big data* en la nube.

5. Aprendizaje autónomo y visualización de datos

5.1. Introducción

En todo sistema de *big data*, además de la captura, el almacenamiento y el procesado de datos, son fundamentales las funciones de:

1) Aprendizaje sobre los datos para retroalimentar los sistemas de procesado de datos.
2) Visualización de datos.

Describimos los principios básicos de estas dos funcionalidades, y las técnicas disponibles, en las siguientes secciones.

5.2. Aprendizaje autónomo: *machine learning*

El **aprendizaje autónomo** (también conocido por su denominación en inglés, ***machine learning***) es el conjunto de métodos y algoritmos que permiten a una máquina aprender de manera automática a partir de experiencias pasadas.

Generalmente, un algoritmo de aprendizaje autónomo debe construir un modelo a partir de un conjunto de datos de entrada que representan el conjunto de aprendizaje, lo que se conoce como conjunto de entrenamiento. Durante esta fase de aprendizaje, el algoritmo va comparando la salida de los modelos en construcción con la salida ideal que deberían tener estos modelos, para ir ajustándolos y aumentando la precisión. Esta comparación forma la base del aprendizaje en sí, y este aprendizaje puede ser supervisado o no supervisado.

El aprendizaje puede ser:

1) **Supervisado**: hay un componente externo que compara los datos obtenidos por el modelo con los datos esperados por este, y proporciona retroalimentación al modelo para que vaya ajustándose. Para ello, pues, será necesario proporcionar al modelo un conjunto de datos de entrenamiento que contenga tanto los datos de entrada como la salida esperada para cada uno de esos datos.

2) **No supervisado**: el algoritmo de entrenamiento aprende sobre los propios datos de entrada, y descubre y agrupa patrones, características, correlaciones, etc.

Algunos algoritmos de aprendizaje autónomo requieren un gran conjunto de datos de entrada para poder converger hacia un modelo preciso y fiable, y el *big data* es un entorno ideal para este aprendizaje.

Tanto Hadoop (con su paquete de aprendizaje autónomo, Mahout) como Spark (con su solución MLlib) proporcionan un extenso conjunto de algoritmos de aprendizaje autónomo.

Algunos tipos de aprendizaje son:

- **La regresión**: se trata de un tipo de aprendizaje supervisado cuyo objetivo es, dado un conjunto de aprendizaje con datos de entrada y sus respectivas salidas, tratar de encontrar una línea (recta o curva) que los aproxime de tal manera que, en el futuro, podamos predecir las salidas a partir de nuevas entradas. Una regresión no pretende devolver una predicción exacta sobre un evento futuro, sino una aproximación. Por lo general, datos más dispersos resultarán en predicciones menos ajustadas.

- **Clasificación**: es otro tipo de aprendizaje supervisado, estrechamente relacionado con las regresiones y en el que, dado un punto en un espacio multidimensional, se pretende decidir qué partes de ese espacio pertenecen a los elementos de un grupo de clases. Al contrario que las regresiones, que pretendían modelar un conjunto de datos como un continuo, la salida de los algoritmos de clasificación está discretizada a un conjunto de clases.

- **Agrupamiento**: el agrupamiento es un tipo de clasificación no supervisada en la que, dado un conjunto de elementos, se pretende clasificarlos en función de unas características representadas como puntos en un espacio multidimensional.

- **Reducción de dimensionalidad**: se basa en la reducción del número de variables que se vayan a tratar en un espacio multidimensional. Este proceso ayuda a reducir la aleatoriedad y el ruido a la hora de llevar a cabo otras operaciones de aprendizaje autónomo, puesto que se eliminan aquellas dimensiones que tengan escasa correlación con los resultados que hay que estudiar. En ocasiones, reduciendo la dimensionalidad no solo se ahorrará en cálculos a la hora de llevar a cabo regresiones o clasificaciones, sino que incluso estas serán más precisas, al reducir ruidos del conjunto de entrenamiento.

- **Filtrado colaborativo / sistemas de recomendación**: tratan de predecir los gustos/intenciones de un usuario a partir de sus valoraciones previas y las valoraciones de otros usuarios con gustos similares. Son muy usados en sistemas de recomendación de música, películas, libros y tiendas en línea. Cuando se crea un perfil de gustos del usuario, se hace utilizando dos formas o métodos en la recolección de características: implícitas o explícitas. Formas explícitas podrían ser solicitar al usuario que evalúe un objeto o tema particular, o mostrarle varios temas/objetos y que elija su preferido. Formas implícitas serían tales como guardar un registro de artículos que el usuario ha visitado en una tienda, canciones que ha escuchado, etc.

5.3. Análisis de grafos

Con el auge del *big data*, la teoría de grafos goza de gran popularidad para el análisis de redes sociales, análisis de grandes cantidades de documentos enlazados entre sí (como las páginas web), o búsquedas de caminos y optimización, entre otras cosas.

La teoría de grafos modela entidades (representadas como vértices) y sus relaciones (representadas como aristas que unen vértices), y permite resolver distintos problemas del área de las ciencias, la ingeniería, la economía y muchos ámbitos no relacionados con el *big data*, como por ejemplo control de producción, redes de ordenadores, automatización, etc.

Tanto Hadoop como Spark proporcionan elementos para procesar grafos: Giraph y GraphX, respectivamente.

5.4. Visualización del *big data*

La visualización de datos es un campo esencial dentro del análisis estadístico, ya que se trata de la base de dos pasos importantes dentro del proceso de análisis exploratorio de datos:

1) Facilita al científico de datos llevar a cabo el análisis exploratorio de los conjuntos de datos, de tal manera que pueda estimar características y proponer hipótesis sobre estas, como paso previo al análisis de datos mediante otros modelos y técnicas más precisas.

2) Permite la exposición de las conclusiones derivadas, y respalda el proceso de comprensión y/o posterior decisión. Para conseguir este fin, el propio creador de la visualización debe tener muy clara la información que desea comunicar, para que le resulte más sencillo transmitírsela a otras personas a través de las observaciones y los cotejos oportunos de los datos constituyentes de la visualización.

La visualización es en sí una herramienta que da apoyo al análisis estadístico, no una técnica analítica o estadística. Los usuarios la utilizan para llevar a cabo comparaciones o determinar causalidades. Facilita el análisis exploratorio y la comunicación de conclusiones, pero no sustituye las técnicas estadísticas usadas para extraer valor de los datos.

Los buenos gráficos estadísticos muestran datos complejos con claridad, precisión y eficiencia.

Algunas de las características de una buena visualización son las siguientes:

- Muestran los datos relevantes.

- Inducen a quien los ve a pensar acerca de la información subyacente más que en la metodología, diseño, tecnología o cualquier otro aspecto técnico, evitando distorsionar qué quieren decir los datos.

- Muestran mucha información en poco espacio.

- Son coherentes con grandes conjuntos de datos.

- Incentivan la comparación de diferentes partes de los datos.

- Muestran distintos niveles de detalle, desde una vista *grosso modo* a pequeños detalles.

- Cumplen un propósito claro: describir, explorar, tabular, etc.

- Están integrados con las descripciones estadísticas y verbales de un conjunto de datos.

- Deben aportar la precisión de los cálculos estadísticos convencionales, y ser más reveladores que estos.

- Están adaptados a los conocimientos de la audiencia a la que se destinan.

- Son autocontenidos (siguen aportando valor incluso fuera de un contexto más grande, como un informe).

La no aplicación de los principios descritos puede llevar a gráficos engañosos y poco orientativos, que distorsionan el mensaje o refuerzan una conclusión errónea. Por ejemplo, sobrecargar los gráficos de decoraciones que no mejoran el mensaje, o efectos gratuitos tridimensionales o de perspectiva.

Las formas y técnicas de visualizar los datos son muy distintas, y con la informática y las herramientas para el análisis *big data*, se van incorporando otros avances técnicos tales como animaciones, interactividad, navegación, etc.

Algunas de las formas más clásicas de representar datos son:

- **Tablas**, que muestran números o clases de elementos. Su comprensión a primera vista se puede mejorar con colores, tonos, leyendas, explicaciones previas, etc.

- **Gráfico de barras**, para poner énfasis en la comparación entre elementos.

- **Gráfico de línea**, para mostrar las relaciones de los cambios en los datos en un periodo de tiempo.

- **Diagrama de cajas**, gráfico que está basado en cuartiles y mediante el cual se visualiza la distribución de un conjunto de datos. Está compuesto por un rectángulo, la caja, y dos brazos, los bigotes. Es un gráfico que suministra información sobre los valores mínimo y máximo, los cuartiles Q1, Q2 o mediana y Q3, y sobre la existencia de valores atípicos y la simetría de la distribución.

- **Gráfico circular**, también conocido como pastel. Se utiliza para mostrar cómo diferentes partes representan un total.

- **Diagrama de dispersión**, también conocido como *scatter plot* en inglés. Son útiles para mostrar la relación entre diferentes puntos de datos.

- **Gráfico de burbujas**, variación de un diagrama de dispersión en el que los puntos de datos son reemplazados por burbujas y el tamaño de las burbujas representa una dimensión adicional de los datos.

- **Mapa de árbol**, conocido como *tree map* en inglés, muestra datos jerárquicos en forma de rectángulos que ocupan el total del espacio de forma proporcional al valor de una variable.

- **Gráficos sociales**, un mapa que muestra cómo se relacionan entre sí las personas u otras entidades. Estos gráficos constan de nodos y flechas (relaciones) que conectan los nodos.

- **Cartograma**, mapa o diagrama que muestra datos de cantidad asociados a sus áreas respectivas, mediante la modificación de los tamaños de las unidades de enumeración.

- **Mapa de densidad**, mapa que muestra la distribución a través de un área geográfica de una variable que hay que medir, generalmente asociada a un color o textura.

De entre los mensajes cuantitativos que pueden ayudar a comunicar o entender un conjunto de datos, podemos encontrar, por ejemplo:

- **Series temporales**, que muestran las mediciones de una variable a lo largo de un periodo de tiempo (por ejemplo, la tasa de desempleo durante un periodo de 20 años). Se pueden utilizar líneas o barras para mostrar la tendencia.

- ***Rankings***, en los que se miden subdivisiones categóricas. Se muestran en un orden ascendente o descendente (por ejemplo, los productos más vendidos de una tienda) durante un periodo. Se puede usar un gráfico de barras para mostrar las comparaciones entre elementos.

- **Parte a todo**, en los que las categorías se miden como una proporción del total (por ejemplo, un porcentaje). Puede usarse un gráfico circular o un gráfico con múltiples barras.

- **Distribución de frecuencia**, que muestra las observaciones de una variable particular para un intervalo de frecuencia dado. Por ejemplo, qué porcentaje de la población tiene una edad comprendida de 0 a 9 años, de 10 a 20, de 20 a 30, etc. Un histograma, mostrado como un gráfico de barras, ayuda a visualizar las estadísticas sobre la distribución. Puede ayudar también un diagrama de caja que muestre datos adicionales sobre la distribución, mediana, media, desviación estándar, etc.

- **Correlación**, que es la comparación entre las observaciones representadas por dos variables, que determine si estas se mueven en la misma (o en opuesta) dirección. Por ejemplo, superponer la tasa de crecimiento del producto interior bruto de un país con la tasa de desempleo puede mostrar la relación entre estas dos variables. Pueden usarse, solos o superpuestos, diagramas de líneas, de barras, de dispersión, etc. Es importante tener en cuenta que correlación no implica relación causa-efecto.

- **Comparación nominal**, que compara subdivisiones categóricas sin un orden particular. Por ejemplo, el volumen de ventas según un código de producto cualquiera. Un gráfico de barras es adecuado para este tipo de gráfico.

- **Geográfico o geoespacial**, que es la comparación de una variable a lo largo de un mapa o distribución del espacio, como por ejemplo la tasa de desempleo por provincia. Típicamente, se utilizan cartogramas o mapas de densidad.

5.5. Herramientas para la visualización de datos

Existen un gran número de herramientas para la visualización de datos. Estas se pueden clasificar de la manera siguiente:

1) De código abierto, como las herramientas Atlasboard, D2 Data Driven Documents, Gephi, Kibana, Matplotlib, Nodebox, NVD3 o R.

2) Soluciones propietarias de pago, como Highcharts, IBM Cognos y Watson Analytics, Periscope Data, QlikView, Quadrigram, SAS Visual Analytics, Tablas Dinámicas de Google, Tableau, o Visually, entre otras muchas.

Resumen

En este material, hemos introducido los principales componentes de una arquitectura *big data*. Hemos empezado hablando de cómo almacenar los datos en sistemas de archivos distribuidos, así como de las ventajas que el almacenamiento distribuido ofrece. No hay duda de que dadas sus bondades, se trata de un sistema muy apropiado para las soluciones inteligentes de la industria 4.0.

Posteriormente, hemos introducido tres técnicas principales para el procesado de datos en grandes cantidades y a gran velocidad:

1) En modo *batch*.

2) En modo *streaming*, para tiempo real.

3) A partir de eventos complejos (CEP).

Finalmente, hemos presentando algunas de las soluciones comerciales disponibles tanto para el procesamiento de datos, como para su análisis y visualización.

Las opciones son amplias, y en este material no hemos profundizado en todas ellas. Sin embargo, lo importante es conocer la existencia y necesidad de los elementos de cualquier sistema de *big data*, y su necesidad para implementar cualquier aplicación que pueda explotar los datos para aportar valor a la industria 4.0.