Caso de uso de *big data*

Análisis de la puntualidad de las compañías aéreas en el período Octubre 1987 – Abril 2008

Josep Curto

PID_00237656

Universitat Oberta de Catalunya

Índice

Introducción

Este caso práctico presenta un caso de uso de *big data* y consta de tres partes:

1) Enunciado del caso. En él se presenta caso de uso "Análisis de la puntualidad de las compañías aéreas en el período Octubre 1987 - Abril 2008" y el enfoque de la actividad.

2) Solución. Se presenta una forma de abordar este proyecto y se detallan cada una de las fases.

3) Anexo. Se proporcionan detalles sobre la configuración en local de este proyecto para su replicación.

Objetivos

Este material didáctico está dirigido a los siguientes profesionales:

- Desarrolladores y consultores que quieren conocer cómo se implementa un proyecto de *big data*.
- Desarrolladores y consultores que quieren ayudar al desarrollo de estrategias de negocio que incluyan *big data*.
- Gestores que están interesados en la transformación digital de su organización y en la inclusión de *big data* como uno de sus pilares fundamentales.

Por otro lado, tiene los siguientes objetivos:

- Entender cuáles son las fases de un proyecto de *big data*.
- Conocer cómo desarrollar cada una de las fases del proyecto.
- Introducir algunas de las tecnologías actuales en proyectos *big data*. En este caso particular, Apache Spark, Apache Zeppelin y Docker.

Si bien la obra es autocontenida en la medida de lo posible, los conocimientos previos necesarios son los siguientes:

- Conocimientos básicos sobre *big data*.

En este módulo se introducirán los conceptos necesarios para el seguimiento del material.

1. Enunciado del caso

La puntualidad es uno de los aspectos más relevantes en el contexto de las compañías aéreas. No solo desde la perspectiva del cliente, que sufre los problemas de puntualidad, sino también desde la perspectiva de la compañía aérea, puesto que este problema supone un impacto en la marca, en los costes y en el beneficio. Por lo que conocer, e incluso predecir qué vuelos son más puntuales y qué factores afectan principalmente a la puntualidad puede ayudar a mejorar los servicios de las compañías en este sector.

Por ello en 2009, la American Statistical Society propuso el reto de entender el rendimiento de las diferentes compañías aéreas en Estados Unidos en el período comprendido entre 1987 y 2008.

En este caso nos vamos a poner en la piel de un investigador interesado en entender este conjunto de datos para poder generar conocimiento a partir del mismo. Eso sí, por los años transcurridos podremos apalancarnos en las tecnologías de *big data* para la captura, el procesamiento, el análisis y la visualización de datos.

Lectura complementaria

J. Curto (2016). *Fundamentos de big data*. Barcelona: UOC

1.1. El reto

Siguiendo el objetivo original, se busca proporcionar un resumen gráfico de las características importantes del conjunto de datos vinculado con la puntualidad de las compañías aéreas. Algunas de las preguntas propuestas que nos interesan analizar son las siguientes:

1) ¿Cuándo es el mejor momento del día para volar minimizando los retrasos? ¿Y del día de la semana? ¿Y de la época del año?

2) ¿Los aviones mayores sufren más retrasos?

3) ¿Cómo ha evolucionado el número de pasajeros por aeropuerto a lo largo del período de análisis?

4) ¿Son las condiciones meteorológicas un buen indicador del retraso de un avión?

5) ¿Produce el retraso en un aeropuerto un retraso en cascada en otros aeropuertos?

6) ¿Hay enlaces críticos en el sistema de aeropuertos?

Por lo tanto, se intuye que el conjunto de datos puede aportar valor para lo que se quiere analizar, pero se deja abierta la puerta a responder a diferentes análisis.

1.2. El conjunto de datos

El conjunto de datos consiste en las salidas y llegadas de todos los vuelos comerciales dentro de Estados Unidos, desde octubre 1987 hasta abril de 2008. En este conjunto de datos tenemos aproximadamente 120 millones de registros y ocupan 1,6 GB en formato comprimido y 12 GB en formato no comprimido. Están disponible para su descarga desde la página web de Stat-Computing.

La descripción del significado de los campos puede encontrarse en la página web de Transtats.

También existe información complementaria que es posible usar para enriquecer el análisis: información de los aeropuertos, de las compañías, de los aviones o incluso de las condiciones meteorológicas.

2. Solución del caso

En el momento de encarar un proyecto de *big data* es necesario seguir un proceso sistemático. Existen pasos comunes a todos los proyectos que están vinculados con las tareas que hay que realizar: adquisición del dato (*Data Acquisition*), almacenamiento del dato (*Data Storage*) o análisis de dato (*Data Analysis*).

En este caso de uso de *big data* vamos a presentar una serie de pasos necesarios para llevar a buen puerto este tipo de proyectos y que además permiten estructurar la solución:

1) Evaluación del caso de negocio: donde se identifica y se comprende el caso de negocio.

2) Identificación de datos: donde se identifican las fuentes de datos relevantes para el caso de negocio.

3) Adquisición y filtrado de datos: donde se adquieren las fuentes de datos y se filtran en función de su finalidad.

> Por ejemplo, en el caso de aplicar técnicas de *machine learning* separaríamos el conjunto de datos en dos: un conjunto para crear el modelo y el otro para testearlo.

4) Extracción de datos: donde se diseñan, implementan y automatizan las técnicas de extracción de datos.

5) Validación de datos: donde se valida el nivel de calidad y la gobernanza de los datos asociados al caso de uso

6) Agregación y representación de datos: donde se agregan y se estructuran los datos en función de las necesidades de análisis. Por ejemplo, en forma de grafo o agregaciones a nivel mensual.

7) Análisis de datos: donde se aplican técnicas de análisis de datos, desde el análisis descriptivo al análisis preventivo.

8) Visualización de datos: donde se diseñan e implementan técnicas de visualización de datos para presentar los resultados.

9) Uso de los resultados: donde se toman decisiones fundamentadas en los resultados del proyecto. Esto puede estar embebido en un producto o servicio de datos o formar parte de un proceso de toma de decisiones.

2.1. Evaluación del caso de negocio

Tal y como se plantea en el enunciado, el origen de este caso de uso proviene de una competición de carácter abierto respecto de un conjunto de datos. Existe un objetivo inicial, que es el análisis de la puntualidad, pero no existen unos objetivos de negocio.

Este caso de uso es mucho más normal de lo esperado. Las organizaciones suelen tener acceso a conjuntos de datos que no han sido explotados, bien porque no existían las tecnologías adecuadas o bien porque los gestores no habían puesto la atención en los mismos.

Bajo estas premisas este es un escenario abierto de análisis. Y podemos considerarlo como un escenario de **validación de hipótesis y resolución de problemas**. Nuestro objetivo, por lo tanto, será explorar el peso de los factores presentes en el conjunto de datos y validar su relevancia.

Es decir, analizaremos la información desde distintos puntos de vista para intentar identificar cuan relevantes son los distintos factores que influyen en los vuelos (zona geográfica de origen y destino, meteorología, horas del día, compañías aéreas, etc.) y ver qué conclusiones podemos sacar de ello.

2.2. Identificación de datos

Aunque los datos de este caso los encontramos, tal y como se comenta en el enunciado, en la página web de stat-computing.org, la fuente original es la página web de la Research and Innovative Technology Administration (RITA).

El conjunto de datos es una versión reducida del conjunto original de RITA, donde se han eliminado ciertos campos calculados y que presenta la información separada por años.

El contenido de cada CSV sigue la estructura que se presenta la tabla 1 recogida de la página web de la propuesta (en el idioma original puesto que los conjuntos de datos están en inglés).

Tabla 1. Descripción de variables

Número	Nombre	Descripción
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1(Monday)-7(Sunday)
5	DepTime	actualdeparturetime(local,hhmm)

Número	Nombre	Descripción
6	CRSDepTime	scheduleddeparturetime(local,hhmm)
7	ArrTime	actualarrivaltime(local,hhmm)
8	CRSArrTime	scheduledarrivaltime(local,hhmm)
9	UniqueCarrier	uniquecarriercode
10	FlightNum	flightnumber
11	TailNum	planetailnumber
12	ActualElapsedTime	inminutes
13	CRSElapsedTime	inminutes
14	AirTime	inminutes
15	ArrDelay	arrivaldelay,inminutes
16	DepDelay	departuredelay,inminutes
17	Origin	originIATAairportcode
18	Dest	destinationIATAairportcode
19	Distance	inmiles
20	TaxiIn	taxiintime,inminutes
21	TaxiOut	taxiouttimeinminutes
22	Cancelled	wastheflightcancelled?
23	CancellationCode	reasonforcancellation(A=carrier,B=weather, C=NAS,D=security)
24	Diverted	1=yes,0=no
25	CarrierDelay	inminutes
26	WeatherDelay	inminutes
27	NASDelay	inminutes
28	SecurityDelay	inminutes
29	LateAircraftDelay	inminutes

Por lo tanto, estamos hablando de 21 ficheros que siguen el mismo patrón estructurado, presentando una serie de perspectivas de las principales métricas y motivos de retraso, como por ejemplo la fecha, la compañía aérea (*carrier*) o el aeropuerto. De los campos destacamos:

1) ***Carrier Delay***: la causa de la cancelación o retraso se ha debido a circunstancias que entran dentro del ámbito de la compañía aérea (por ejemplo, problemas en el mantenimiento o la tripulación, limpieza de las aeronaves, carga de equipajes, repostaje, etc.).

2) ***Weather Delay***: la causa de la cancelación o retraso se ha debido a condiciones climáticas extremas o peligrosas, reales o previstas, en el punto de partida, en la ruta o en el punto de destino.

3) ***NAS Delay***: la causa de la cancelación o retraso es atribuible al Sistema Nacional de Aviación, que se refiere a un amplio conjunto de condiciones, tales como condiciones meteorológicas no extremas, operaciones aeroportuarias, elevado volumen de tráfico o el control del tráfico aéreo.

4) ***Security Delay***: los retrasos o cancelaciones son causados por la evacuación de una terminal, el reembarque de aviones a causa de una violación de la seguridad o a la inoperatividad y/o espera excesiva en los escáneres de acceso.

5) ***Late Aircraft Delay***: la causa de la cancelación o retraso se ha debido a que un vuelo anterior operado con el mismo avión llega tarde, causando que el presente vuelo salga también con retraso.

Por lo que tenemos las siguientes condiciones:

- Datos estructurados.

- Datos con un volumen creciente, que en primera instancia son 12 GB, pero que potencialmente puede llegar a cantidades superiores si se consideran años y meses posteriores a abril 2008.

Según estas condiciones, tenemos un escenario de escalabilidad horizontal así como volumen, lo que según la clasificación de NIST tenemos un escenario de *big data* tipo 2.

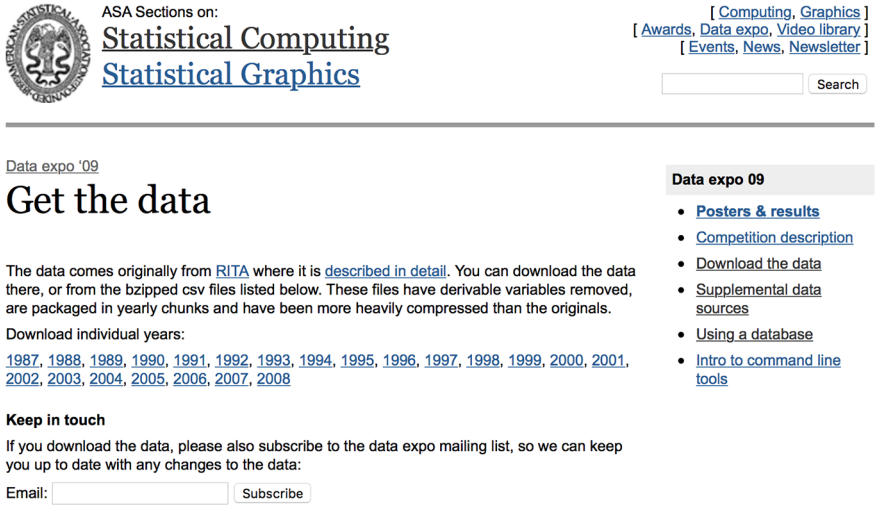
Por lo tanto, teniendo en cuenta las fuentes de datos, su tipología, uso exploratorio y tipo, y donde no importa el tiempo real, en el momento de seleccionar una tecnología necesitamos una plataforma con foco en *batch processing*.

Para este caso práctico vamos a considerar la plataforma Apache Spark por sus capacidades para trabajar con *batch processing*.

2.3. Adquisición y filtrado de datos

En este caso, la adquisición de datos se realiza desde la página web indicada en el enunciado del caso, como es posible ver en la figura 1.

Figura 1. Recuperando los datos



Fuente: Josep Curto

Debemos descargar cada uno de los conjuntos de datos. Así mismo, puesto que están comprimidos es necesario extraer los ficheros. Se recomienda descomprimirlos todos dentro de una carpeta. Durante el proceso de preparación del entorno combinaremos esta carpeta con la plataforma de *big data*.

2.4. Extracción de datos

El primer paso tras conseguir los datos es extraerlos en la plataforma que estamos usando. Para ello necesitamos una plataforma. Vamos a usar un entorno de desarrollo fácil y rápido de desplegar en cualquier sistema operativo. En el subapartado 4.3 se explica cómo configurar Apache Spark con Apache Zeppelin (como entorno de trabajo) mediante Docker. Se recomienda su lectura antes de proseguir.

Una vez ya hemos preparado el entorno podemos empezar con la extracción de los datos. Siguiendo los pasos del subapartado 4.3 tendremos los diferentes ficheros descomprimidos.

El uso de Zeppelin facilita la combinación de diferentes lenguajes para cada una de las etapas posteriores. Podemos activar diferentes intérpretes como Markdown para la composición del texto, Spark para el acceso a las capacidades de procesamiento y almacenamiento de Apache Spark o SQL para hacer consultas SQL.

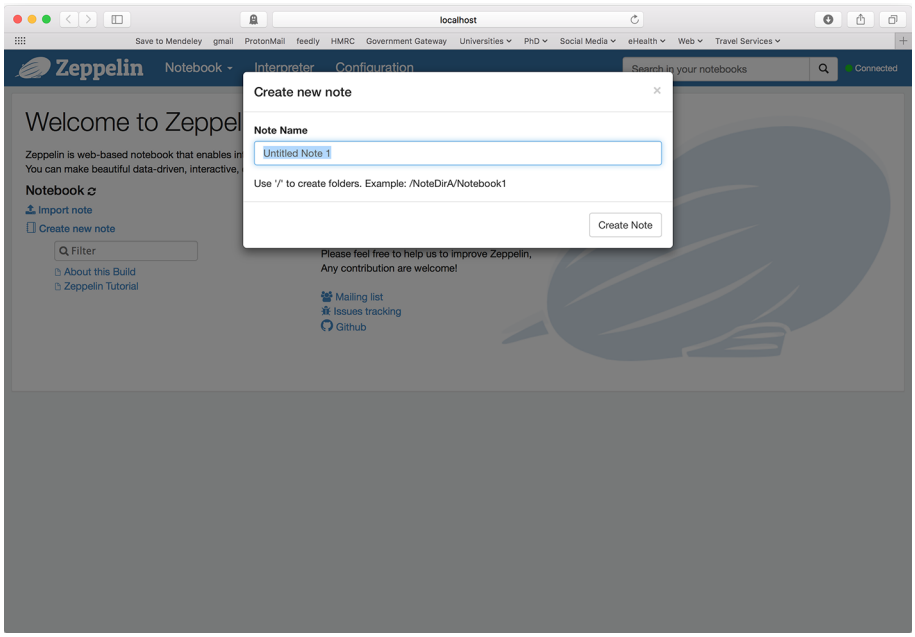
Markdown

Cuando hablamos de Markdown hacemos referencia al lenguaje de marcado ligero creado por John Gruber en 2004 que permite crear ficheros de texto fácil-de-leer y fácil-de-escribir, y con la posibilidad de poder convertir el documento en XHTML.

Es importante destacar que el uso de plataformas de *big data* implica que deberemos programar el sistema, aunque el uso de Apache Zeppelin minimiza la programación y facilita la exploración en una plataforma de *big data*.

Los pasos necesarios se describen a continuación. En primer lugar, creamos un nuevo *notebook* pulsando en la orden *Create a new notebook* como se ilustra en la figura 2.

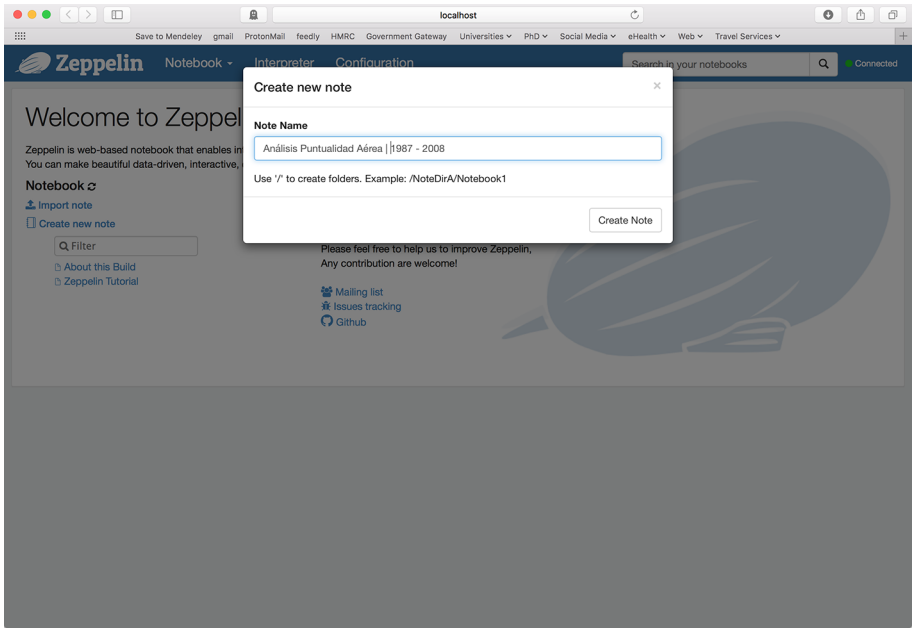
Figura 2. Creando un nuevo *notebook*



Fuente: Josep Curto

A este nuevo *notebook* le damos un nombre coherente vinculado a su análisis (en nuestro caso análisis de puntualidad aérea en el período adecuado) como se ilustra en la figura 3.

Figura 3. Añadiendo nombre al *notebook*.

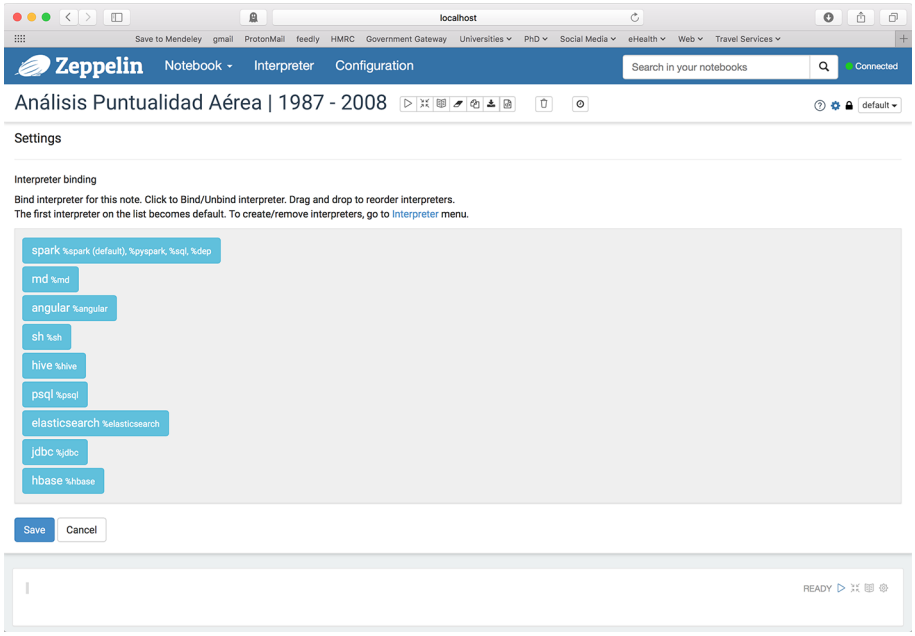


Fuente: Josep Curto

Cuando entramos por primera vez en nuestro *notebook* se nos indican los intérpretes disponibles. Si sabemos que no se usarán ciertos intérpretes podríamos desactivarlos (como *elasticsearch* o *hive*).

Hay también un espacio de trabajo en la parte inferior (que denominamos celda) que usaremos a partir de ahora, como se ilustra en la figura 4.

Figura 4. Pantalla inicio del *notebook*



Fuente: Josep Curto

Ahora es el momento de empezar a trabajar.

Primera celda

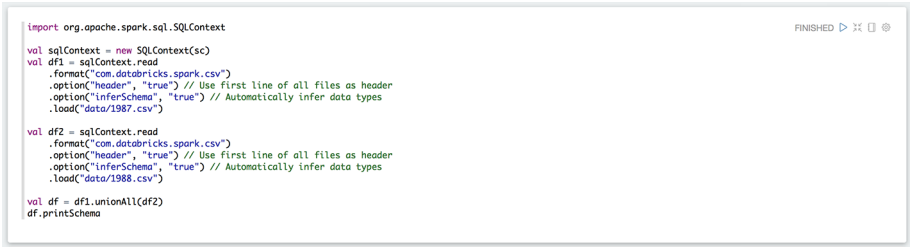
En la primera celda, vamos a cargar los datos, combinarlos y visualizar el esquema del conjunto de datos resultante. Es decir, creamos dos *data frames* (para cargar los datos de los años 1987 y 1988), los combinamos por filas y revisamos el esquema de los CSV en formato árbol. El código usado es el siguiente.

```
import org.apache.spark.sql.SQLContext

val sqlContext = new SQLContext(sc)
val df1 = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("data/1987.csv")

val df2 = sqlContext.read
    .format("com.databricks.spark.csv")
    .option("header", "true")
    .option("inferSchema", "true")
```

```
    .load("data/1988.csv")


val df = df1.unionAll(df2)
df.printSchema
```

Lo escribimos en la celda disponible en el *notebook* como se ilustra en la figura 5.

Figura 5. Completando la primera celda



Fuente: Josep Curto

2.5. Validación de datos

Normalmente es necesario validar el nivel de calidad del dato. En este caso, el conjunto de datos fue preparado para una competición donde el objetivo no está vinculado con la gobernanza, por lo que este paso no es necesario.

Hay un punto interesante en el proceso que hemos seguido. Hemos dejado que el sistema detecte el esquema del fichero y el tipo de cada campo. Sin embargo, al revisar los resultados podemos ver que algunos de los campos no están en el formato correcto, lo que nos va a dificultar el análisis posterior.

Por ello creamos otra celda para poder cambiar el formato de los campos. Lo que hacemos en ella es crear una función para transformar los tipos y luego aplicarlo a varios campos que no estaban bien formateados.

Segunda celda

```
def convertColumn(df: org.apache.spark.sql.DataFrame, name:String, newType:String) = {
   val df_1 = df.withColumnRenamed(name, "swap")
   df_1.withColumn(name, df_1.col("swap").cast(newType)).drop("swap")
}


val df1 = convertColumn(df, "ArrDelay", "int")
val df2 = convertColumn(df1, "DepDelay", "int")
val df = df2
df.printSchema
```

2.6. Agregación y representación de datos

Nuestro objetivo es explorar el conjunto de datos en su nivel máximo de detalle. Para ello creamos también una tabla temporal, a la que llamamos *ontime*, que nos permitirá hacer consultas SQL sobre la misma y explorar el nuevo conjunto de datos.

Tercera celda

```
df.registerTempTable("ontime")
```

En este primer paso hemos realizado la ingesta del dato, cierto procesamiento y su carga en memoria en el sistema. Esto nos va a permitir a continuación hacer ciertos análisis.

Lo natural sería hacer persistente este dato para su posterior análisis.

> Por ejemplo, persistir la tabla creada en formato HDFS para poder acceder ya a los datos desde esta capa en lugar de los ficheros originales. Incluso es posible plantear hacer persistente el dato en bases de datos de baja latencia como Apache Cassandra, aunque ello sólo es posible si se modifica el contenedor de Docker con el que estamos trabajando.

En este caso no se va a realizar la persistencia del dato.

2.7. Análisis y visualización de datos

Al usar Apache Zeppelin estamos combinando al mismo tiempo el análisis y la visualización de los datos. Esto significa que cuando ejecutamos una consulta o aplicamos un algoritmo a la estructura de datos, podemos ver automáticamente el resultado. Esto nos facilita el trabajo. En otras plataformas y con otras configuraciones estas dos fases están separadas en diferentes herramientas.

Cuarta celda

Tras haber creado el objeto tabla, podemos hacer una consulta para ver todos los elementos. El entorno de consultas automáticamente nos proporciona una visualización en formato tabla y diversos tipos de gráficos que pueden aplicarse.

En esta primera consulta el formato tabla es el adecuado para analizar la información accedida. Existen dos métodos. O bien llamando al intérprete SQL:

```
% sql
select
   *
From
   ontime
```

O bien directamente accediendo al contexto:

```
val results =sqlContext.sql("select * from ontime")
results.show()
```

El primer método genera unos gráficos y tablas para el análisis; el segundo, genera objetos que podemos reusar a posteriori.

Quinta celda

Podemos empezar a analizar los datos; por ejemplo, recuperando el número de vuelos que estamos analizando en los años 1987 y 1988.

A través del intérprete:

```
% sql
select
   Year, count(1) as cantidad
from
   ontime
group by
   Year
order by
   Year
```

A través del contexto (es necesario poner toda la consulta en una única línea):

```
val results =sqlContext.sql("select
Year, count(1) as cantidad
from
   ontime
group by
   Year
order by
   Year")
results.show()
```

Sexta Celda

También podemos preguntarnos qué ha pasado en las llegadas y en las salidas por año y compañía aérea. A través del intérprete:

```
% sql
select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year, UniqueCarrier
from
   ontime
```

```
group by
   Year, UniqueCarrier
```

A través del contexto (es necesario poner toda la consulta en una única línea):

```
val results =sqlContext.sql("select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year, UniqueCarrier
from
   ontime
group by
   Year, UniqueCarrier")
results.show()
```

Séptima Celda

Y continuar con qué ha pasado en las llegadas y en las salidas por ruta y año. Una ruta se compone de un aeropuerto de salida y otro de llegada. A través del intérprete:

```
% sql
select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year, Origin, Dest
from
   ontime
group by
   Year, Origin, Dest
```

A través del contexto (es necesario poner toda la consulta en una única línea):

```
val results =sqlContext.sql("select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year, UniqueCarrier
from
   ontime
group by
   Year, UniqueCarrier")
results.show()
```

Octava Celda

Nos preguntamos qué tipo de retraso ha sido más importante. A través del intérprete:

```
% sql
select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year
from
   ontime
```

```
group by
   Year
```

A través del contexto (es necesario poner toda la consulta en una única línea):

```
val results =sqlContext.sql("select
   sum(ArrDelay) as Arrival, sum(DepDelay) as Departure, Year
from
   ontime
group by
   Year")
results.show()
```

2.8. Uso de los resultados

Este breve análisis nos ha permitido empezar a comprender el conjunto de datos. Tan solo cargando dos de los ficheros ya tenemos los siguientes resultados:

- Disponemos de un conjunto de 1.311.826 vuelos durante el año 1987 y 5.202.096 durante el año 1988.

- United Airlines (UA) es la compañía que acumula más retrasos en 1988 tanto en la salida como en la llegada, seguida por American Airlines (AA).

- La ruta San Francisco (SFA) - Los Ángeles (LAX) es la ruta que más retrasos acumula en la llegada.

3. Actividades

En este caso práctico hemos empezado el análisis exploratorio de los datos de puntualidad. Es necesario extender este análisis. Este caso práctico se puede mejorar de diversas maneras:

- Mejorando el proceso para que no sea necesario descomprimir los ficheros.
- Cargando el resto de años y combinandolos con los anteriores en un único *data frame* para analizar todos ellos de forma conjunta.
- Combinando el conjunto de datos de retrasos con los conjuntos de datos complementarios, como por ejemplo los nombres de aeropuertos, para trabajar con un conjunto de datos completo.
- Actualizando el análisis presentado para el nuevo conjunto de datos.
- Creando una capa de persistencia de datos.
- Realizando una comparación por décadas.
- Realizando una comparación por compañías aéreas.
- Realizando una comparación por aeropuertos.
- Realizando una comparación por rutas aérea.
- Determinando cómo han evolucionado los factores de retraso a lo largo del tiempo.
- Analizando mediante regresión u otras técnicas los factores de retraso.
- Actualizando el caso con los datos provenientes de RITA.

4. Anexo

4.1. Apache Spark

En esta actividad se ha considerado Apache Spark como el ecosistema de uso para *big data*.

Apache Spark es una plataforma *open source* para el procesamiento de datos que nace para superar las limitaciones de MapReduce basado en el paradigma *Resilient Distributed Datasets* (RDD). Estas limitaciones consisten en que MapReduce fuerza a seguir un proceso lineal de lectura, mapeo, reducción y almacenamiento del dato en disco, lo que supone unas latencias de escritura y lectura. Spark supera esta limitación a partir del uso de la memoria (distribuida) del clúster en lugar de la escritura a disco.

Lectura complementaria

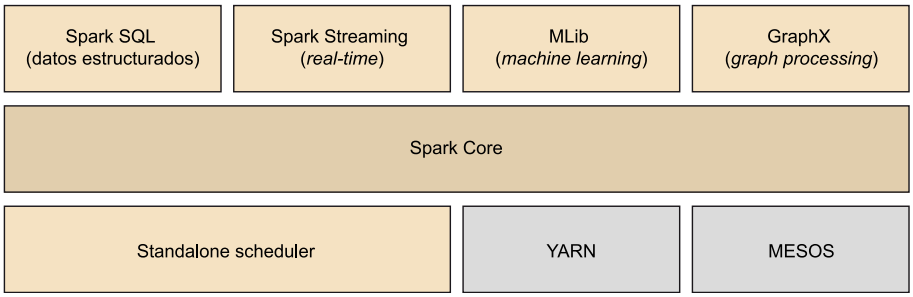
M. Zaharia; M. Chowdhury; T. Das; A. Dave; J. Ma; M. McCauley; M. J. Franklin; S. Shenker; I. Stoica (2012). *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. Berkeley: University of California.

El origen de Spark podemos encontrarlo en los desarrollos de Matei Zaharia[1] en 2009 cuando estaba en UC Berkeley AMPLab. En 2010 pasó a convertirse en *open source*. Spark soporta escenarios de *batch processing* y *streaming processing* por encima del segundo.

[(1)]Matei Zaharia es el actual CTO de Databricks.

El ecosistema de Apache Spark se estructura como se ilustra en la figura 6.

Figura 6. Ecosistema Apache Spark



Fuente: Apache Spark

El ecosistema de Spark incluye componentes para la gestión como YARN y MESOS, un núcleo de procesamiento y varios componentes de procesamiento de datos estructurados, *streaming*, *machine learning* y grafos. La comunidad de Spark es bastante activa y está desarrollando múltiples conectores para extender el ecosistema presentado.

4.2. Docker

Docker es una herramienta que permite crear aplicaciones empaquetadas auto-eficientes, llamadas contenedores. Docker aparece como evolución de las imágenes virtuales para mejorar el uso de los recursos de un servidor o computadora.

Estas aplicaciones empaquetadas se caracterizan, por lo tanto, por ser muy livianas (en términos de consumo) y que son capaces de funcionar en prácticamente cualquier ambiente (sistema operativo), ya que contiene todas las dependencias necesarias para su funcionamiento (código, *frameworks*, herramientas, librerías, archivos, configuraciones, etc.).

En esencia, Docker permite tener exactamente la misma configuración y funcionalidad en los diferentes entornos de desarrollo, calidad y producción.

4.3. Configurando el entorno

Para hacer la actividad se ha usado un Docker ya existente que contiene Spark y Zeppelin y que está alojado en un repositorio público en hub.docker.com creado por Josep Curto (https://hub.docker.com/r/josepcurto/sparkds/).

Puesto que hemos identificado que el caso de uso es validación de hipótesis y resolución de problemas, necesitamos un entorno fácil de desplegar y configurar.

Este Docker está basado en Debian y contiene:

1) Apache Spark 2.0.0

2) Apache Hadoop 2.7.0

3) Soporte para PySpark con Python 3.4, NumPy y SciPy

4) Diversos intérpretes instalados (lo que significa que soporta): Spark, Shell, Angular, Markdown, PostgreSQL, JDBC, Hive, HBase y Elasticsearch.

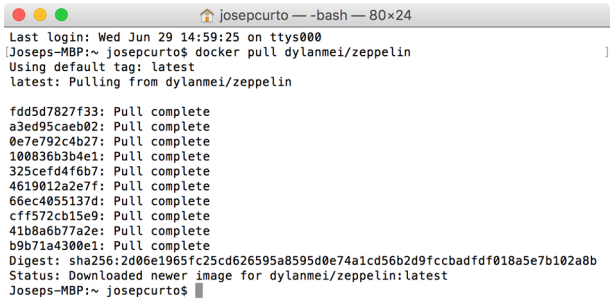
5) Apache Zeppelin 0.7.0

Los pasos para preparar el entorno son:

1) Instalar Docker para el correspondiente sistema operativo.

2) Abrir un terminal y descargarse la última versión del contenedor usando la orden: `docker pull josepcurto/sparkds`. Esta orden permite recuperar un contenedor ya creado por un desarrollador como se ilustra en la figura 7.

Figura 7. Recuperando el contenedor de Spark con Zeppelin



Fuente: Josep Curto

3) Debemos indicar una ruta para poder guardar los datos. Nos interesan que los datos sean externos a la imagen para que sean persistentes y recuperables. En nuestro caso `/Users/josepcurto/Downloads/zeppelin` y en esta carpeta creamos la carpeta datos donde guardaremos los ficheros csv descomprimidos.

4) Ejecutamos la orden que inicia el contenedor así como mapea los volúmenes:

```
docker run -rm -name zeppelin -p 8080:8080 -v /Users/josepcurto/Downloads/
zeppelin/datos:/usr/zeppelin/data josepcurto/sparkds
```

5) Para acceder al contenedor (trabajando a través de Apache Zeppelin), abrimos el navegador y escribimos "http://docker_host:8080" donde *docker_host* hace referencia a la IP donde se aloja el contenedor. En esta actividad, *localhost*.

6) Finalmente debemos ir a la sección *Interpreter* y editar el intérprete de Spark añadiendo la dependencia `com.databricks:spark-csv_2.11:1.4.0`, tal como se ilustra en la figura 8.

Figura 8. Añadiendo el CSV *interpreter*

Fuente: Josep Curto

4.4. Órdenes útiles en Docker

En este subapartado recopilamos varias órdenes relevantes para trabajar con Docker. Estas órdenes deben ejecutarse en una terminal.

Problema	Órdenes
¿Cómo saber qué versión y toda la información de Docker?	docker info
¿Cómo saber qué imágenes tenemos en Docker?	docker images
¿Cómo conocer el detalle de las imágenes en Docker?	docker ps

Problema	Órdenes
¿Cómo borrar todas las imágenes y contenedores en Docker?	docker rm $(docker ps -a -q) docker rmi $(docker images -q)
¿Cómo conocer la estructura de carpetas de una imagen específica?	docker inspect $(nombre_imagen)
¿Cómo parar una imagen encendida?	docker stop $(nombre_imagen)

Resumen

En este caso práctico hemos presentado cómo desarrollar un primer proyecto de validación de hipótesis en la forma de un piloto para *big data*.

Hemos aprendido cómo evaluar la problemática de negocio, cómo analizar las necesidades de negocio, cómo vincularlas a una tecnología específica y cómo generar valor de negocio.

Además hemos introducido dos tecnologías: Apache Spark, como ecosistema de *big data* seleccionado para la problemática analizada, y Docker, como herramienta de desarrollo.

Finalmente hemos discutido las potenciales siguientes fases de este proyecto.

Glosario

big data *m* Conjunto de estrategias, tecnologías y sistemas para el almacenamiento, procesamiento, análisis y visualización de datos complejos.

byte *m* Unidad de medida de información digital.

escalabilidad *f* Habilidad de un sistema, red o proceso para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

escalabilidad horizontal *f* Escalabilidad fundamentada en el incremento de nodos del sistema, proceso o red.

escalabilidad vertical *f* Escalabilidad fundamentada en añadir más recursos (memoria, disco duro y/o procesadores).

latencia *f* Suma de retardos temporales en la captura, almacenamiento, procesamiento y análisis del dato.

metadatos *m pl* Datos estructurados y codificados que describen características de un objeto, dato o proceso de negocio.

variabilidad *f* Característica de los flujos de datos que supone que pueden tener comportamientos erráticos o inconsistentes en ciertos períodos.

variedad *f* Se refiere tanto a la cantidad de fuentes diferentes que combinar como a la heterogeneidad del dato.

velocidad *f* Se refiere tanto al procesamiento de datos como a su latencia.

veracidad *f* Incertidumbre en el dato producto de su baja calidad, la ambigüedad en su definición o las simplificaciones en su modelización.

vinculación *f* Dificultad de relacionar diferentes y dispares fuentes de datos.

volumen *m* Tamaño del conjunto de datos creado diariamente.

Bibliografía

Davenport, T. H.; Harris, J. G. (2007). *Competing on Analytics: The New Science of Winning*. Nueva York: Harvard Business Press.

Davenport, T. H.; Kim, J. (2013). *Keeping Up with the Quants: Your Guide to Understanding and Using Analytics*. Boston: Harvard Business Review Press.

Davenport, T. H. (2014). *Big Data at Work: Dispelling the Myths, Uncovering the Opportunities*. Boston: Harvard Business Review Press.

Erl, T.; Khattak, W.; Buhler. P. (2015). *Big Data Fundamentals: Concepts, Drivers & Techniques*. New Jersey: Prentice Hall

Fisher, T. (2009). *The Data Asset: How Smart Companies Govern Their Data for Business Success*. New Jersey: Wiley

Foreman, J. W. (2013). *Data Smart: Using Data Science to Transform Information into Insight*. New Jersey: Wiley.

Malcolm, F.; Roehrig, P.; Pring, B. (2014). *Code Halos: How the Digital Lives of People, Things, and Organizations Are Changing the Rules of Business*. New Jersey: Wiley

Redman, T. C. (2008). *Data Driven: Profiting from Your Most Important Business Asset*. Boston: Harvard Business Review Press.

Schmarzo, B. (2013). *Big Data MBA: Understanding How Data Powers Big Business*. New Jersey: Wiley.

Schmarzo, B. (2016). *Big Data MBA: Driving Business Strategies with Data Science*. New Jersey: Wiley.