Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE GRADO

**Título: "**Análisis, uso y desarrollo experimental de herramientas y tecnologías Open Source en Big Data"

**Titulación**: Grado en Ingeniería Telemática y Grado en Ingeniería de Sistemas de Telecomunicación

**Autores:** Jean-Paul Fannes Claverol
Marc Pallejà Mairena

**Director:** Roc Meseguer Pallarès

**Fecha:** 12 de Septiembre del 2017

**Título: "**Análisis, uso y desarrollo experimental de herramientas y tecnologías Open Source en Big Data"

**Titulación**: Grado en Ingeniería Telemática y Grado en Ingeniería de Sistemas de Telecomunicación

**Autores:** Jean-Paul Fannes Claverol
Marc Pallejà Mairena

**Director:** Roc Meseguer Pallarès

**Fecha:** 12 de Septiembre del 2017

## Resumen

En este trabajo se pretende analizar, usar y justificar la utilización del Big Data hoy en día en diferentes ámbitos, como empresas, lugares de investigación, etc., así como las herramientas y tecnologías Open Source que hay detrás que lo hacen posible.

La metodología utilizada para este proyecto, ha consistido en realizar un análisis exhaustivo del presente y futuro de Big Data, la presentación de herramientas y tecnologías Open Source disponibles, el estudio y comparativa de cada una de ellas y el desarrollo experimental final utilizando datos propios.

Destacamos sobre todo el uso de Big Data, con las tecnologías y herramientas asociadas como Hadoop y Spark, como una gran alternativa de almacenamiento y procesado de elevados volúmenes de datos.

Ya que se trata de un concepto nuevo para nosotros, vamos a tratar de explotar al máximo todas las funcionalidades Open Source relacionadas con Big Data, y poder dar una recomendación personal sobre cada una de ellas evaluando las principales características.

Finalmente, profundizaremos un poco más sobre el concepto de Machine Learning en el que hoy muchas empresas focalizan sus principales desarrollos de tecnologías. Para ello, haremos un pequeño desarrollo experimental con el principal objetivo de localizar a personas que estén conectadas a través de sus dispositivos a diferentes access points en un área concreta. El objetivo principal es estudiar diferentes algoritmos para obtener la mayor precisión posible de localización de la persona y comparar los diferentes resultados obtenidos, a través del procesado y análisis de datos conseguidos como Edificio, Planta, etc.

**Palabras clave**
*Big Data, Apache Hadoop, Apache Spark, Lenguaje R, Scala, Machine Learning, Tecnologías y herramientas de Big Data, Comparativa.*

**Title:** “Analysis, use and experimental development of Open Source tools and technologies in Big Data”

**Degree:** Bachelor degree in Network Engineering and Bachelor degree in Telecommunication Systems Engineering

**Authors:** Jean-Paul Fannes Claverol
Marc Pallejà Mairena

**Director:** Roc Meseguer Pallarès

**Date:** September 12th 2017

## Overview

In this project, we pretend to analise, use and justify the use of Big Data nowadays in different areas like companies, research laboratories, etc., as well as different tools and Open Source technologies behind it that make it possible.

The methodology used for this project has consisted on doing an exhaustive analysis of the current and future situation of Big Data, the introduction of the tools and Open Source technologies available, the study and comparative of each of them and the final experimental development with our own data.

We highlight the use of Big Data, along with the associated technologies and tools such as Hadoop and Spark, as a great alternative to classic way of storing and processing big amounts of data.

Given that it is a new concept for us, we are going to try to exploit as much as possible the Open Source functionalities related to Big Data and we are going to give our own personal recommendation on each one evaluating their main characteristics.

Finally, we are going to dig further on the concept of Machine Learning on which a lot of companies are focusing their development. In order to do this, we are going to do an experimental project aiming to detect people’s position based on the access points whose devices are connected to. The main objective is to study different algorithms to obtain the best possible accuracy of personal position and to compare the different results throw the analysis and processing of obtained data such as Building, Floor, etc.

**Key Words**
*Big Data, Apache Hadoop, Apache Spark, R language, Scala language, Machine Learning, Big Data technologies and tools, Comparative.*

# AGRADECIMIENTOS

Nos gustaría agradecer a todas aquellas personas que nos han apoyado en el transcurso de este proyecto, ya sea aportando ideas de mejora en temas de contenido o diferentes sugerencias gramaticales, haciendo el proyecto más rico en todos los aspectos.

Queremos destacar y agradecer la figura de nuestro tutor y director de proyecto (Roc Meseguer), ya que en todo momento ha mostrado un gran interés y disposición a ayudarnos en todo lo referente a este trabajo, además de mostrar también una gran profesionalidad y comprensión con nosotros desde el primer momento que empezamos a realizarlo.

Finalmente, queremos dar las gracias a nuestros familiares por darnos su apoyo incondicional y por todo el soporte recibido y ánimos que nos han proporcionado desde que empezamos a trabajar en este proyecto, ya que, en los principales momentos de dudas y agobio nos han estimulado en todo momento. Aunque ha sido un proceso duro, creemos que ha merecido la pena el esfuerzo del trabajo realizado para finalizar con éxito nuestros respectivos grados de Ingeniería.

# ÍNDICE

# INTRODUCCIÓN

El trabajo de final de grado que presentamos a continuación se divide en 5 grandes capítulos, los cuáles están ordenados progresivamente sobre conceptos, herramientas y tecnologías de Big Data y finalmente el uso de algunas de estas sobre una base datos para ejecutar una pequeña aplicación.

A modo de resumen general explicaremos brevemente la temática de los capítulos:

El primer capítulo será una presentación general del Big Data y porque es tan importante hoy en día, su definición teórica, una breve historia que hay detrás y la actualidad y tendencias acerca del tema. Además, haremos una breve introducción al capítulo 2 nombrando los principales lenguajes de programación, tecnologías presentes y entornos de desarrollo acotando todo el escenario global.

En el segundo explicaremos de forma teórica las herramientas y tecnologías Open Source presentes (algunas empresas disponen de su propio software) con tal de dar una idea general de lo que podemos encontrar y también dar una visión introductoria al lector.

En el capitulo tres utilizaremos dichas tecnologías para ver, mediante el uso de ejemplos, las principales ventajas e inconvenientes de cada una, haciendo énfasis en la gran cantidad de posibilidades que nos pueden ofrecer. Compararemos procesos de instalación y uso de librerías, entornos de programación y lenguajes, respuestas en forma de tiempo del mismo código adaptado para ver el proceso de datos, etc. En definitiva, haremos una estrecha comparación de todas las tecnologías y herramientas, y haremos una conclusión final de cuáles son los aspectos más destacados de cada una.

En el cuarto, después de la comparación, escogeremos las herramientas y tecnologías que más nos gusten y profundizaremos en ellas. Concretamente queremos utilizar Machine Learning para calcular la posición de los dispositivos que estén conectados a un punto de acceso de Wi-Fi, para ello utilizaremos diferentes evaluaciones, ajustándolas a los algoritmos que más nos convengan y modificando diferentes opciones para acercarnos al resultado deseado.

Por último, en el capítulo cinco, acabaremos con las conclusiones finales del proyecto y haciendo una reflexión general sobre todo lo aprendido sobre el mundo de Big Data.

# CAPÍTULO 1. PRESENTACIÓN TEÓRICA Y ANÁLISIS DE LA ACTUALIDAD DE BIG DATA

## 1.1 – Motivación

Al pensar que tema queríamos hacer de TFG, estuvimos barajando diferentes opciones con un denominador común, que fuera totalmente de actualidad y en pleno auge en el sector de las Tecnologías de la Información y Comunicación.

Uno de esos temas a evaluar fue Big Data. Dado que, durante la carrera no vimos absolutamente nada, es algo que nos llamó mucho la atención ya que muchas empresas hablaban de su importancia a causa de la infinidad de datos que se llegan a recoger por la red, sobre todo ahora que casi todos los dispositivos tienen una conexión a Internet. Big Data es un concepto de actualidad que siempre aparece en las noticias tecnológicas de actualidad, además, muchas ofertas de trabajo ahora se centran en la búsqueda de personas para el análisis exhaustivo y masivo de datos.

Descartando las otras opciones, nos centramos en Big Data ya que queríamos aprender sobre las herramientas y tecnologías que se usan y cómo se aplican. Es un campo muy demandado en los sectores tecnológicos, y asentar las bases conceptuales a la vez que hacemos nuestros primeros acercamientos prácticos, puede resultar muy interesante para nuestro futuro como Ingenieros.

## 1.2 – Objetivos principales

Los principales objetivos que nos proponíamos, y que nos hemos ido imponiendo, durante realización de este proyecto son los siguientes:

- Investigar que es Big Data, de que trata este concepto y todo lo que hay detrás.

- Estudiar y usar las diferentes tecnologías, herramientas Open Source y lenguajes de programación presentes hoy en día relacionados con Big Data. Realizar un análisis individual y comparativo, además de establecer una pequeña guía introductoria para gente que tenga conocimientos sobre programación, pero no sobre Big Data.

- Familiarizarnos con los diferentes entornos de desarrollo asociados, ya sea sobre propios sistemas operativos como Ubuntu 16.04 LTS o entornos virtualizados como Cloudera, además de adentrarnos en la programación con lenguajes desconocidos para nosotros como Scala o R.

- En la parte final haremos un proyecto real con una herramienta o tecnología concreta. Dicho proyecto consistirá en predecir las posiciones de diferentes dispositivos conectados a Access Points visibles mediante algoritmos de Machine Learning. Posteriormente se evaluará el potencial que puede tener la herramienta de elección y un proyecto como el nuestro.

## 1.3 – Planificación de actividades

La planificación de las actividades en este proyecto tiene una gran importancia. Al no ser individual hay que organizarse lo mejor posible el tiempo disponible, para ello hemos utilizado la herramienta gratuita Trello, para dividirnos en forma de semanas todas las tareas y poder hacer un seguimiento exhaustivo de todo lo que hemos ido realizando.

Dichas tareas se iban asignando individualmente o conjuntamente dependiendo de su dificultad y el reparto de actividades para cada uno, e intentando cumplir una fecha de finalización más o menos coherente para no ir atrasados.

## 1.4 – Concepto de Big Data

Big Data se podría definir como el conjunto de herramientas informáticas destinadas a la manipulación, gestión, análisis, almacenamiento, visualización y consultas de grandes volúmenes de datos de todo tipo mucho mayores a los que podemos tratar con técnicas tradicionales.

El objetivo fundamental es dotar de una infraestructura tecnológica a las empresas y organizaciones con la finalidad de procesar la gran cantidad de datos que se generan diariamente para aprovecharlas; se requiere tanto hardware como software específico para gestionarlos y obtener la información útil.

## 1.5 – Historia breve previa al Big Data

- Comienzo de la era computacional (1930-1949)
  - Turing y Good descodifican mensajes cifrados alemanes en la 2ª GM.
  - El Predictor Kerrison automatiza la defensa antiaérea.
  - El Proyecto Manhattan realiza simulaciones para predecir comportamientos nucleares.

- Comercialización de la analítica (1950-1969)
  - Ordenador ENIAC hace la primera predicción meteorológica.
  - La analítica resuelve el “problema del camino más corto”.
  - FICO aplica modelos predictivos a diferentes decisiones.

- Análisis popularizado (1970-1999)
  - Modelo Black-Sholes para predecir el precio óptimo de las acciones.
  - Lanzamiento de Amazon y eBay.
  - Google aplica algoritmos a las búsquedas web para maximizar resultados.

## 1.6 – Actualidad y tendencias del Big Data

Sin darnos cuenta usamos o vemos aplicaciones a diario que se nutren del Big Data, las recomendaciones personalizadas que nos hacen webs de compras como Amazon, motores de búsqueda que manejan cientos de direcciones, como Google, o los gráficos publicados sobre tráfico o contaminación lumínica en ciudades son algunos de los ejemplos.

El almacenaje de datos ha ido aumentando a lo largo de los años, además de adaptarse a las nuevas tecnologías para su posterior análisis por parte de las empresas. El volumen de datos de tráfico IP de consumidores y usuarios sigue en alza en todo el mundo. Por ejemplo, en 2021, se espera que el tráfico IP global alcance el valor de casi 233 Exabytes al mes con una tasa de crecimiento anual del 24 por ciento. Dicho tráfico en 2016 obtuvo un valor de 78.250 PB y va aumentando año a año, además, el auge de las comunicaciones móviles (que veremos en el Gráfico 1.2) colabora mucho al aumento del tráfico.

En el Gráfico 1.1 podemos ver plasmado lo que acabamos de comentar:

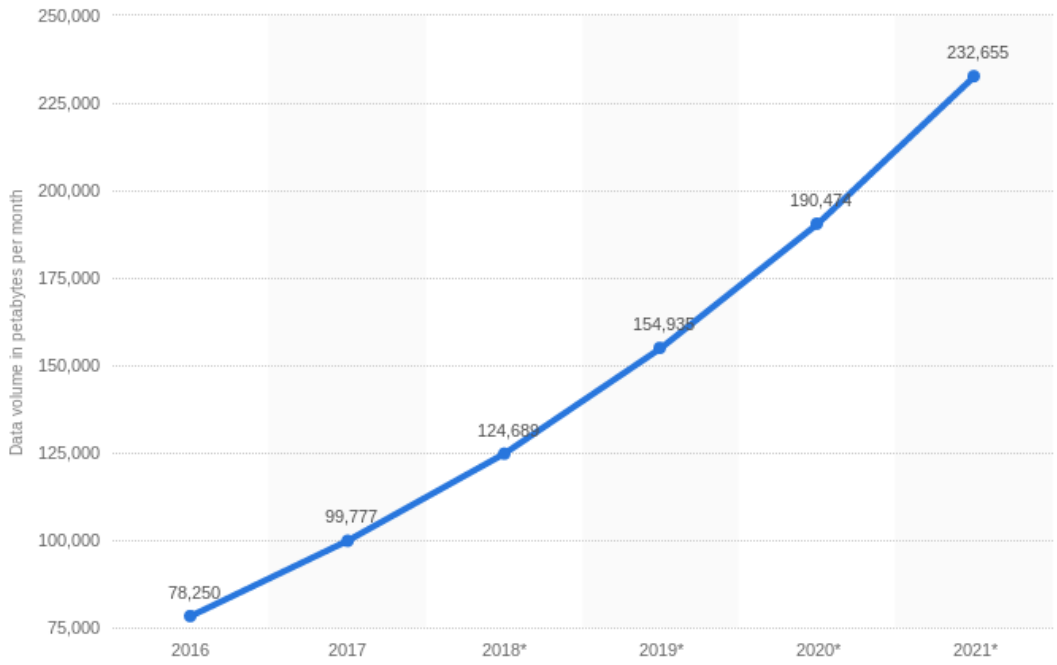



**Gráfico 1.1**– Volumen global de tráfico de datos IP por mes y año (2016-2021)

También podemos observar mediante los estudios de las VNI (Visual Networking Index), de la empresa Cisco, el aumento enorme de la generación de datos gracias al avance inminente de la tecnología y al estilo de vida. Estos estudios

representan múltiples publicaciones periódicas para evaluar el impacto y el auge del crecimiento de datos en las redes móviles y poder actuar en consecuencia.

En esta parte, la VNI del 2016 hizo diferentes estudios y predicciones sobre los próximos años hasta 2021, como podemos observar en el Gráfico 1.2.

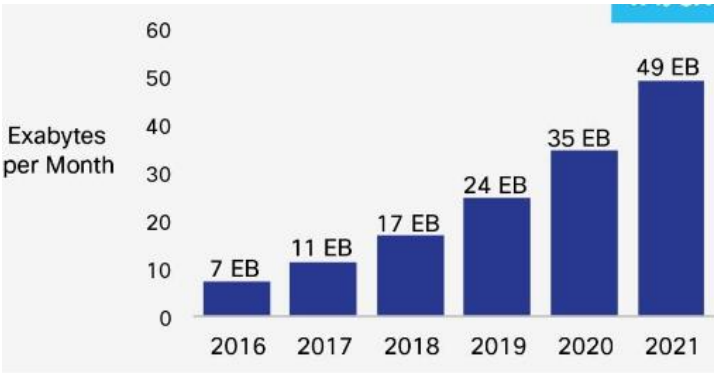



**Gráfico 1.2** – Tráfico de datos móviles global en Exabytes por Año (2016-2021)

Observamos que en 2016 se generaban 7 EB por mes, por contra, en 2021 a través de la predicción, se generarán 49 EB por mes, un aumento muy considerable en tan solo 5 años.

Referenciado en el Gráfico 1.3 vemos como las tasas de crecimiento de datos móviles globales varían ampliamente de una región a otra, siendo Oriente Medio y África las que presentan un mayor crecimiento (96%), seguidas por Asia Pacífico (71%), América Latina (66%) y Europa Central y Oriental (64%).

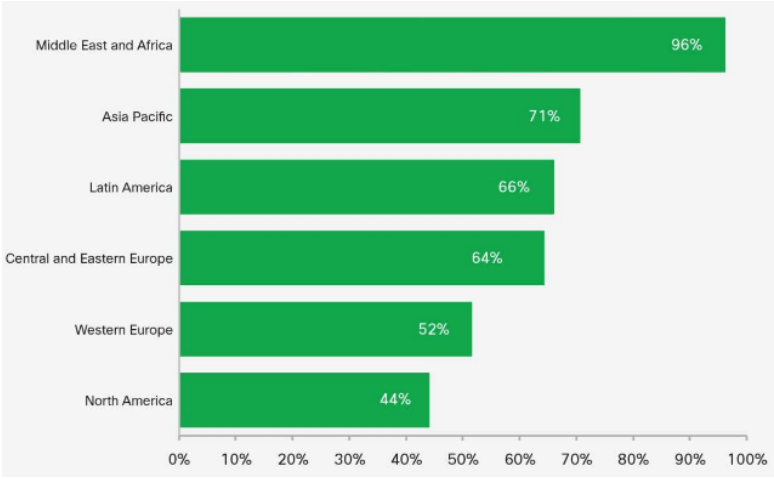



**Gráfico 1.3** – Crecimiento de datos móviles globales por zonas y regiones

Finalmente, el gran crecimiento de los dispositivos inalámbricos que acceden a redes móviles también propicia un gran crecimiento. Globalmente, los dispositivos móviles y conexiones crecerán a 11.6 millones en 2021 tal y como podemos observar en el Gráfico 1.4.

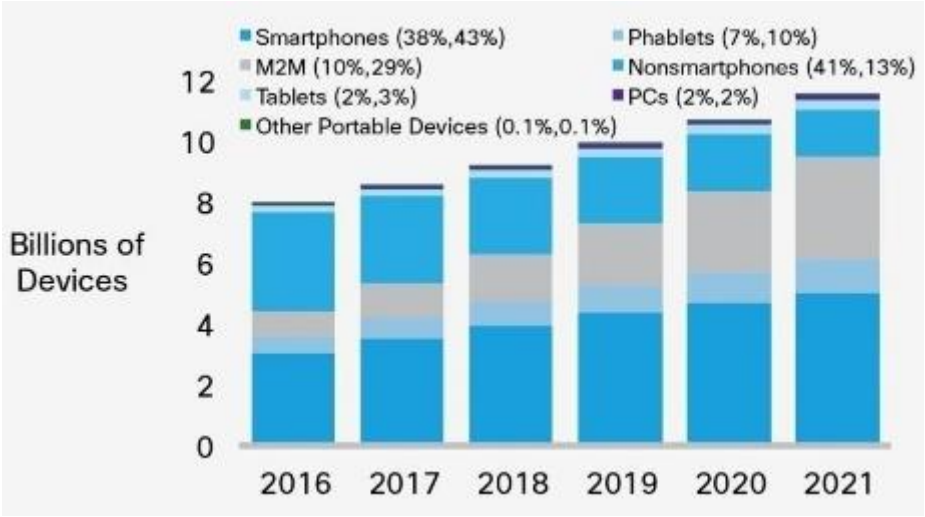



**Gráfico 1.4** – Porcentaje y tipo de dispositivos que utilizan el tráfico de datos global

El término Big Data es muy simple (literalmtente, gran cantidad de datos) pero no lo es todo lo que engloba. Los grandes volúmenes de datos que hemos visto, así como la gran conectividad que existe entre todos los dispositivos, hacen que podamos acceder a todo tipo de datos posibilitando todo un mar de posibilidades. Además, los conceptos como Smart Cities, Internet of Things o Industria 4.0 hacen que todo esté mucho más conectado a Internet y que, por lo tanto, podamos recoger datos de casi todos los dispositivos imaginables.

Creemos que hoy en día es muy importante y un tema de capital importancia, es por ello, que hemos decidido estudiar las principales tecnologías y herramientas que se utilizan y las ventajas y características que podemos sacar de cada una de ellas.

## 1.7 – Punto de partida

En esta primera fase introductoria, el objetivo principal es presentar al lector de una forma general, el marco en el que se encuentra Big Data actualmente, es decir, que tecnologías, lenguajes, plataformas y entornos de desarrollo se utilizan para su uso.

### 1.7.1 – Lenguajes de programación y tecnologías

Empezaremos haciendo un repaso sobre los lenguajes de programación más utilizados en el ámbito de Big Data, enumerándolos y haciendo alguna pequeña mención destacada, ya que en otros capítulos entraremos más en profundidad sobre los que finalmente elijamos.

- Java: orientado a objetos y uno de los lenguajes más utilizados actualmente, gracias a la gran versatilidad que tiene por poder usarse en muchas plataformas.

- Scala: basado en Java, con algunas diferencias y mucho más compacto al utilizar sus funciones, hacen de él una ventaja para simplificar el código en diferentes líneas.
- R: orientado a estadísticas y poca complejidad de uso. Manipulación de datos y gráficos resultantes interesantes.
- SAS: análisis estadístico y herramienta para transformar información de bases de datos a formatos como HTML o PDF, además de tablas y gráficos.
- Python: gran variedad de bibliotecas y funciones estadísticas entre las grandes virtudes que ofrece. Además de estar incorporado ya por defecto en distribuciones GNU/Linux.
- SQL: lenguaje para realizar las bases de datos, manejar los datos y poder utilizar diferentes funciones.
- NoSQL: aparece con la llegada de la web 2.0 y los ejemplos más utilizados son MongoDB y Cassandra entre otros.

Las principales tecnologías utilizadas a día de hoy en Big Data, en cuanto a manejo de clústers, sistemas de archivos y datos, son:

- Apache Hadoop
- Apache Spark
- Apache Storm

Apache Hadoop fue con el que se inició todo, pero a falta de herramientas externas que añadieran más funcionalidades, se creó Apache Spark que proporciona muchas más librerías y más opciones. Veremos esos 2 en este proyecto y nos centraremos en este último.
Apache Storm no lo veremos, pero realiza las funciones en tiempo real de Hadoop (que las hace en Batch).

### 1.7.2 – Plataformas y entornos de desarrollo

Las plataformas y entornos de desarrollo presentes en Big Data son muchas y variadas. Cualquier tecnología o herramienta es totalmente compatible con cualquier sistema operativo, ya sea Windows, Mac OS X o cualquier distribución de GNU/Linux, así que se puede elegir cualquier opción ya que funcionará perfectamente y podremos realizar la instalación sin ningún tipo de problema.

Además, hay que destacar entornos virtualizados creados por diferentes empresas, donde ya se encuentran todas las herramientas y tecnologías de Big Data instaladas por defecto, ahorrándonos el tiempo de instalación y ofreciéndonos apoyo y soporte por si hay algún tipo de problema. Dichos entornos son utilizados por multitud de empresas gracias a su gran comodidad y efectividad, algunos de los más utilizados son Cloudera y Hortonworks.

## 1.8 – Uso en el mundo empresarial

A medida que pasan los años, la importancia de utilizar Big Data va aumentando exponencialmente en el mundo empresarial. Empresas como Amazon, aparte de ofrecer los servicios básicos de compra online, tiene servicios en la nube y servicios de Big Data, estos últimos radican en algoritmos de Machine Learning, que detectan patrones en datos existentes y que utilizan modelos para procesar datos nuevos y generar predicciones.

Otras empresas que contribuyeron firmemente en el desarrollo de tecnologías de Big Data como Google, también han abierto cuota de mercado con dichos servicios para analizar y utilizar los datos. Entre las diferentes características, destaca la información administrada sin servidor, consultas rápidas con petabytes de datos, utilización de Spark y Hadoop en la nube, etc.

Una manera muy fácil de ver el impacto que tiene Big Data en el marco empresarial actual, es buscando ofertas de trabajo para “Data Scientist”. Este término describe al profesional que se dedica a procesar los datos que recauda la empresa o incluso datos externos que puedan ser de interesantes y extraer conclusiones de ellos; en todas las ofertas piden conocimientos de lenguajes y aplicaciones relacionados con Big Data así que es evidente que es un sector en auge en la industria de hoy en día.

## 1.9 – Acotación del escenario actual – Conclusiones

Haciendo toda esta pequeña introducción en este capítulo y viendo las diferencias opciones que tenemos tanto de lenguajes de programación, tecnologías y entornos, vamos a elegir los que nos parecen que pueden dar más juego y que son los más utilizados por multitud de empresas. Con ellos trabajaremos y haremos un análisis exhaustivo evaluándolos y viendo sus características.

- Lenguajes de programación escogidos: Java, Scala, SQL y R.
- Tecnologías escogidas: Apache Hadoop y Apache Spark.
- Entornos de desarrollo: Ubuntu 16.04 LTS para Spark y Cloudera para Hadoop.

# CAPÍTULO 2. TECNOLOGÍAS Y LENGUAJES DE PROGRAMACIÓN A ESTUDIAR

En este capítulo vamos a ver las tecnologías y lenguajes que nos han parecido más interesantes desde un punto de vista teórico. Veremos Apache Hadoop (con Java), Apache Spark (con Scala) y el lenguaje R individualmente.

## 2.1 – Apache Hadoop

En este apartado del trabajo, definiremos y explicaremos las principales características de esta tecnología, la arquitectura básica, conceptos y modos de ejecución relacionados, la utilización y el funcionamiento para sacar conclusiones y poder hacer una comparativa final lo veremos en el siguiente capítulo.

### 2.1.1 - Definición y características

Apache Hadoop es una plataforma que permite el procesamiento de grandes volúmenes de datos a través de un clúster, usando un modelo simple de programación. Está diseñado para escalar de servidores individuales a miles de máquinas, cada una ofreciendo computación y almacenamiento local. Proporciona un framework escrito en lenguaje de programación Java en el cual se pueden desarrollar aplicaciones distribuidas que requieren un uso intensivo de datos y de alta escalabilidad.

El proyecto es licencia de Apache Software Foundation, por tanto, se trata de un software totalmente libre y que permite al usuario que trabaje con él, modificarlo como quiera.

Cabe destacar, que inicialmente el proyecto de Apache Hadoop se inspiró en los documentos de Google para MapReduce y Google File System (GFS) y que ha tenido muchos contribuyentes globales (Facebook, Twitter, LinkedIn...) destacando así Yahoo! como el mayor contribuyente y el que presenta el mayor clúster con dicha tecnología. En aquel momento, Hadoop estaba formado por un sistema de archivos distribuidos denominado HDFS y un modelo de procesamiento y ejecución de datos denominado MapReduce.
Con su progresiva adopción en diversos sectores y en administraciones públicas, Hadoop ha evolucionado con rapidez y se ha convertido en un complemento (y en algunos casos, sustituto) de los almacenes de datos empresariales.

### 2.1.2 - Arquitectura básica, conceptos y modos de ejecución

#### *2.1.2.1 - Arquitectura básica*

La arquitectura básica de Apache Hadoop consta de los siguientes módulos principales (a partir de la versión 2.0):

- Hadoop Common: herramientas comunes que respaldan el resto de módulos de Hadoop, es decir, que posibilitan la integración de otros subproyectos.

- Sistema de ficheros: Hadoop puede acceder a diferentes tipos de sistemas de archivos (local, CloudStore-KFS, Amazon S3, HDFS...). También soporta FTP y HTTP/HTTPS de solo lectura. El Hadoop Distributed File System (HDFS) es el más común y utilizado, se trata de un sistema de archivos distribuidos que almacena datos en máquinas básicas en todo el clúster, en el apartado de *2.1.2.2 HDFS* entraremos en más detalle.

- Hadoop MapReduce: modelo de programación para el procesamiento de datos a gran escala, en el apartado de MapReduce entraremos en más detalle.

- Hadoop YARN: es la plataforma que se encarga de administrar los recursos informáticos de los clústeres y los utiliza para la planificación de las aplicaciones de los usuarios.

Los módulos citados anteriormente son los llamados “módulos básicos”, pero con el paso de posteriores versiones se han ido añadiendo diferentes grupos de herramientas y proyectos totalmente independientes que se pueden instalar en Hadoop para simplificar el acceso y el procesamiento de los datos almacenados en el clúster. Estas son algunas de las más utilizadas, donde haremos una breve explicación:

- Ambari: GUI para administrar y monitorizar clústeres de Hadoop.

- HBase: base de datos escalable que está distribuida y soporta el almacenamiento de datos estructurados en tablas. Realización de tablas a partir de ficheros de datos.

- Hive: infraestructura de almacén de datos con acceso de tipo SQL a los datos.

- Pig: lenguaje de scripting para acceder a los datos y transformarlos.

- Sqoop: gestión del movimiento de los datos entre las bases de datos relacionales y Hadoop.

- Flume: servicio para recopilar datos de archivos de registro en HDFS.

- Mahout: biblioteca de aprendizaje automático.

- Tez: infraestructura de programación de flujo de datos, basada en YARN, para el procesamiento de lotes y las consultas interactivas.

- Zookeeper: servicio de coordinación de alto rendimiento para aplicaciones distribuidas.

- MongoDB Connector for Hadoop: permite utilizar MongoDB como fuente de entrada y destino de tareas de MapReduce, Spark, Hive y Pig.

- Spark: infraestructura de computación de clústeres en memoria utilizada para el procesamiento de lotes rápido, streaming de eventos y consultas interactivas. En el apartado 2.2, hablaremos de él en detalle.

Estas son algunas de las principales herramientas que podemos utilizar junto con Hadoop, finalmente y cómo podemos observar en la Figura 2.1, la principal diferencia entre las versiones 1.0 y 2.0 es que aparece el módulo YARN, así mismo, también se muestra donde se pueden aplicar algunas de las herramientas externas que hemos comentado.

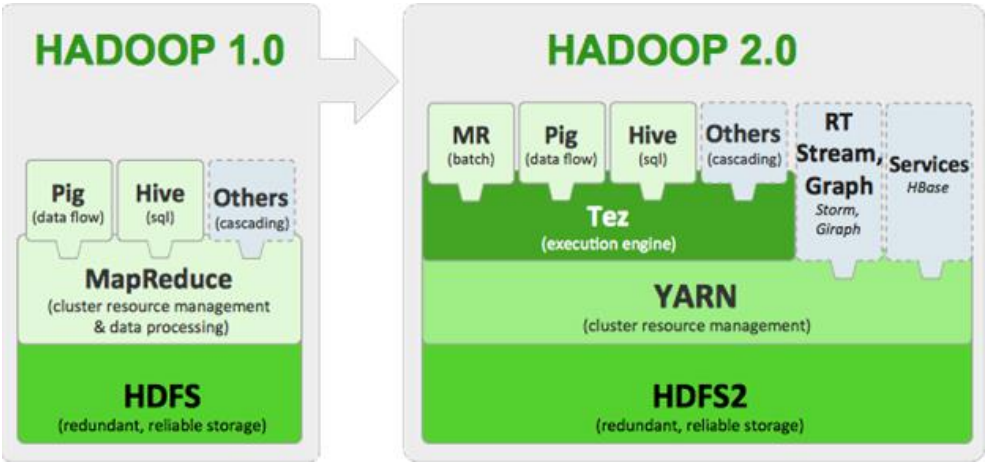



**Figura 2.1** – Comparativa de versiones de Hadoop y uso de herramientas externas

*2.1.2.2 - Conceptos: HDFS*

HDFS es un sistema de archivos distribuido escalable pensado para almacenar grandes cantidades de información y está diseñado para ser instalado en máquinas de bajo coste. Las principales ventajas de HDFS son la tolerancia a fallos (fiabilidad mediante replicación) y la disponibilidad de un elevado ancho de banda. Por el contrario, hay una elevada latencia y es poco eficiente con muchos ficheros de poco tamaño.

La información es dividida en bloques, que son almacenados y replicados en los discos locales de los nodos del clúster. La arquitectura se basa en maestro/esclavo, donde el nodo maestro llamado ***NameNode*** se encarga de manejar el espacio de nombres del sistema, replica los bloques de ficheros y regula el acceso de los clientes a los distintos ficheros indicándoles donde se encuentran, estos ficheros se guardan en los nodos esclavos llamados ***DataNodes***, que se encargan de gestionar el almacenamiento en los discos

locales. Se puede añadir un tercer elemento adicional llamado ***Secondary NameNode*** el cual realizará periódicamente puntos de control sobre los cambios en el sistema de ficheros.

El funcionamiento básico es el siguiente:

- Un cliente debe contactar primero con el ***NameNode*** para indicarle donde se encuentra la información/archivo solicitado.

- El ***NameNode*** retorna el identificador del bloque más relevante y el nodo en el que se encuentra.

- El cliente contacta con el ***DataNode*** para recuperar la información requerida. Toda la transferencia de información se produce directamente entre los clientes y los nodos de datos.

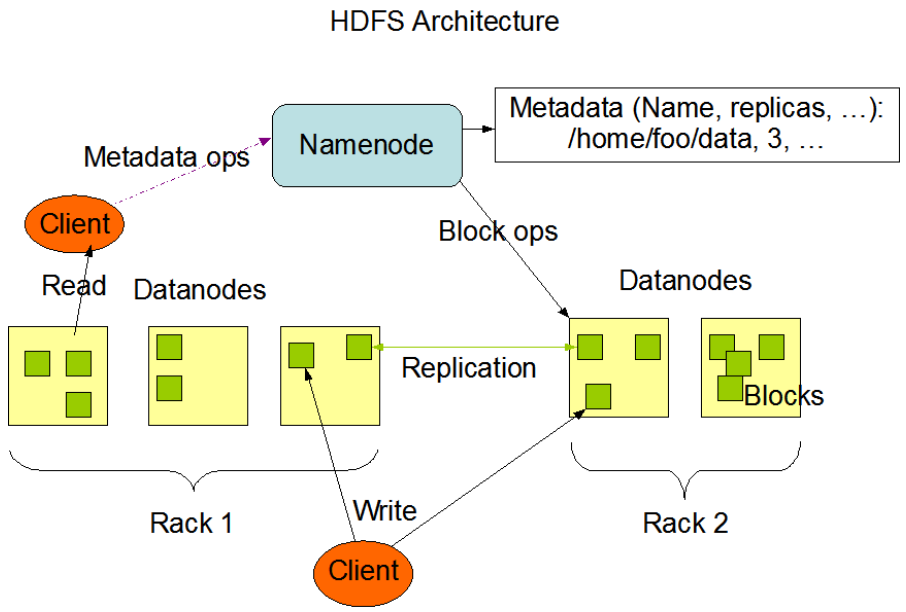



**Figura 2.2** – Arquitectura y funcionamiento básico de HDFS

*2.1.2.3 - Conceptos: YARN*

YARN presenta capacidades de gestión de recursos del clúster que anteriormente residían en MapReduce (que ahora se convierte en una librería de Hadoop) y los empaqueta para que puedan ser utilizados por los nuevos motores de procesado y se puedan ejecutar varias aplicaciones en Hadoop.

YARN provoca que se desarrollen nuevas herramientas para que cubran múltiples necesidades que solo con MapReduce no se podían completar (algunas herramientas ya las hemos comentado en el apartado *2.1.2.1 Arquitectura básica*).

YARN se basa fundamentalmente en separar las dos mayores responsabilidades del ***JobTracker***: la gestión de los recursos y la planificación/monitorización de las tareas en dos servicios separados, a raíz de

eso surgen dos nuevos componentes: el ***ResourceManager*** global y el ***ApplicationMaster***.

El ***ResourceManager*** surge para el master y dictamina los recursos entre todas las aplicaciones del sistema. El ***ApplicationMaster*** negocia los recursos con el ***ResourceManager*** y trabaja con los ***NodeManager*** (que sustituyen al ***TaskTracker*** para cada esclavo) para ejecutar y supervisar las tareas que lo componen. También se añade un componente con el nombre Container que representa los recursos disponibles en cada nodo del clúster.

#### *2.1.2.4 - Conceptos: MapReduce*

MapReduce está orientado para la solución práctica de algunos problemas que pueden ser paralelizados. Presenta una arquitectura maestro/esclavo, con un servidor maestro llamado ***JobTracker*** y varios servidores esclavos llamados ***TaskTrackers***, uno para cada nodo del clúster. Su nombre se debe a las funciones principales que son ***Map*** y ***Reduce***.

La función ***Map*** consiste en coger unos datos y reasignar unos valores de un registro de datos diferente donde cada elemento individual se divide en varios keys(k)/valores(v).

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

**Figura 2.3** – Función Map

La función ***Reduce*** consiste en coger el resultado de la función Map, el nuevo registro de datos basado en keys(k)/valores(v) que agrupa los pares de valores, y realiza una lista de valores nueva.

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

**Figura 2.4** – Función Reduce

Para acabar de explicar su funcionamiento, utilizaremos un ejemplo:

Asumimos que tenemos 5 archivos (en los cuáles no hay datos repetidos) y en el que cada archivo contiene 2 columnas, que representan un equipo de fútbol y sus correspondientes puntos en la liga española durante las diferentes temporadas. En este caso, el equipo de fútbol es la clave (key) y los puntos en la liga es el valor (value). En la Tabla 2.5 podemos ver el contenido de uno de los ficheros.

| Equipo de fútbol (clave - key) | Puntos en La Liga (valor - value) |
|---|---|
| FC Barcelona | 94 |
| Real Madrid CF | 92 |
| Club Atlético de Madrid | 78 |
| Sevilla FC | 63 |
| FC Barcelona | 87 |
| Sevilla FC | 76 |
| Club Atlético de Madrid | 90 |

**Tabla 2.5** – Contenido de un fichero con las dos columnas

De dichos datos presentados en las dos columnas, queremos saber de todas las temporadas de La Liga, cuál es el mayor número de puntos alcanzado por cada equipo. Para realizar dicho proceso, dividiremos la función MapReduce en cinco tareas paralelas, ejecutándose una por archivo. El contenido (Tabla 2.5) de un fichero podría ser este al ejecutarse:

*FC Barcelona, 94; Real Madrid CF, 92; Club Atlético de Madrid, 90; Sevilla FC, 76;*

Ahora revisamos los resultados de los otros cuatro archivos:

*FC Barcelona, 100; Real Madrid CF, 85; Club Atlético de Madrid, 76; Sevilla FC, 50;*
*FC Barcelona, 91; Real Madrid CF, 100; Club Atlético de Madrid, 56; Sevilla FC, 50;*
*FC Barcelona, 99; Real Madrid CF, 96; Club Atlético de Madrid, 47; Sevilla FC, 63;*
*FC Barcelona, 87; Real Madrid CF, 78; Club Atlético de Madrid, 67; Sevilla FC, 70;*

Una vez obtenidos y procesados los 5 ficheros, la función Reduce devolvería:

*FC Barcelona, 100; Real Madrid CF, 100; Club Atlético de Madrid, 90; Sevilla FC, 76;*

Al ver el resultado nos mostraría de todos los ficheros procesados, los equipos con la mayor puntuación obtenida a lo largo de las diferentes temporadas, ya que agruparía cada valor de cada fichero por su clave, con los equipos de fútbol devolviendo una lista. Después con el lenguaje de programación, podríamos realizar cualquier función sobre este resultado.

#### *2.1.2.5 - Modos de ejecución*

Hadoop presenta tres diferentes modos de ejecución, son los siguientes:

- Modo local/stand-alone: por defecto está configurado para ejecutarse en modo no-distribuido como un proceso Java aislado, es decir, que todo está en un nodo. Sirve para entornos de pruebas y depuración.

- Modo pseudo-distribuido: puede ejecutarse en un único modo pseudo-distribuido donde cada demonio se ejecuta en un proceso Java diferente (funciona como una instalación completa, pero en un solo nodo).

- Modo distribuido: forma para aprovechar todo el potencial de esta tecnología, ya que se maximiza el paralelismo de procesos y se utilizan todos los recursos del clúster en el que se va a configurar.

## 2.2 – Apache Spark

Spark fue introducido al público por Apache como una alternativa más veloz a Hadoop, pero Spark no es una versión modificada de Hadoop.
Hadoop es simplemente (en este contexto) una manera de implementar Spark.
Intentaremos repasar esta herramienta de manera similar a Hadoop.

### 2.2.1 - Definición y características

Apache Spark es un framework de código abierto (Open Source) de computación distribuida a través de clústers. Fue desarrollado originalmente en la Universidad de Berkeley (California) y más tarde donado al Apache Software Foundation que desde entonces se encarga de su mantenimiento.

Spark supone una evolución sobre MapReduce (introducido por Hadoop), mantiene su escalabilidad y tolerancia a errores, pero añade otras funcionalidades, por ejemplo, permite programar en Scala, Java, Python, SQL y R.

#### *2.2.1.1 - Velocidad*

Cuando hablamos de Big Data nos referimos a cantidades da datos muy grandes, así que una mayor velocidad significará una mayor interactividad (mas pruebas), la experimentación será más rápida y dará como resultado más productividad. Spark ofrece, sobre el papel, velocidades de hasta 100 veces mayores que Hadoop y esto es posible porque realiza sus operaciones intermedias sobre memoria y no sobre disco.

## 2.2.2 – Arquitectura básica, conceptos y modos de ejecución

Spark es un proyecto diseñado para funcionar con una gran variedad de arquitecturas, además tiene soporte para muchos sistemas de almacenamiento ya existentes, entre ellos todos los que Hadoop ya acepta.

La Figura 2.6 muestra un pequeño diagrama donde vemos el núcleo de Spark, con todas las funcionalidades que se usan aparte del MapReduce y por encima de este, las diferentes librerías que trae por defecto.

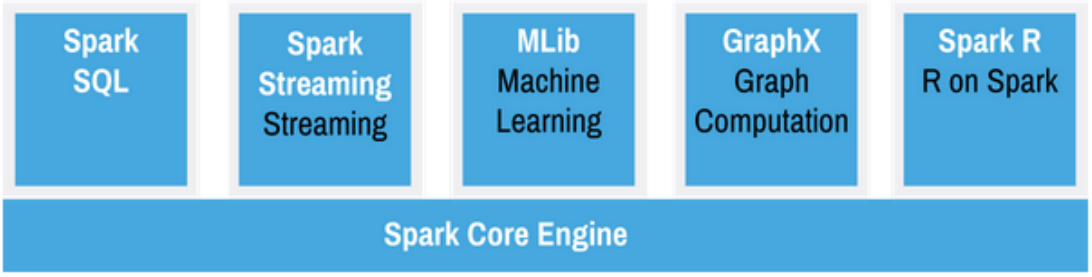



**Figura 2.6** – Diagrama de Spark

### *2.2.2.1 - Librerías externas*

Spark soporta consultas SQL, Streaming de datos, Machine Learning o algoritmos de gráficos. Todo esto lo puede hacer con librerías que vienen ya integradas (MLlib, GraphX, etc.) pero además hay muchos proyectos externos que complementan los de Apache, algunos grandes como “Thunder” una librería que permite analizar datos neuronales a gran escala y otros más pequeños como spark_azure, que proporciona un script para ejecutar Microsoft Azure.

Como hemos comentado una de las características más importantes de Spark es la velocidad de procesado que ofrece, dos conceptos clave para esta velocidad son DAG y RDD.

### *2.2.2.2 – Conceptos: DAG*

DAG es el acrónimo de Gráfica Acíclica Dirigida (Directed Acyclic Graph) y como su nombre indica, para cada nodo del grafo no habrá caminos que empiecen y acaben en sí mismo.

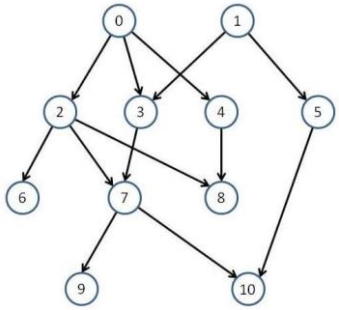



**Figura 2.7** - Ejemplo DAG

Cada tarea de Spark crea un DAG de etapas de trabajo para que sea ejecutado en un determinado clúster. Su antecesor, ***MapReduce***, tiene dos estados predefinidos (***Map*** y ***Reduce***) y para cada una de las etapas tiene que escribir en disco los resultados entre ambas etapas, en contrapartida, los DAG de Spark pueden tener cualquier número de etapas y no tiene que escribir los resultados intermedios.

#### *2.2.2.3 – Conceptos: RDD*

En la mayoría de frameworks el único método de reutilizar datos entre, por ejemplo, dos operaciones de MapReduce, es guardar los datos en un sistema externo (por ejemplo, HDFS).

Spark soluciona esto con los ***RDD*** (Resilient Distributed Dataset), su estructura de datos fundamental, que puede contener cualquier tipo de objeto y mantenerlos en memoria en vez en disco. Sobre los RDD se pueden realizar dos tipos de operaciones, transformaciones, mediantes las cuáles se obtendrá un nuevo RDD tras una transformación, o acciones, que devolverá el resultado de realizar una acción sobre los datos. Mediante la función cache() se pueden almacenar en memoria los datos de un RDD.

Vemos por lo tanto que esto es muy útil al hacer iteraciones o interacciones, y el hecho de no tener que guardar los datos tras cada operación nos proporciona una mayor velocidad de computación.

#### *2.2.2.4 – Modos de ejecución*

A nivel fundamental Spark se puede ejecutar en modo Standalone (en un solo ordenador) o en modo Clúster, donde aprovecha al máximo el concepto de RDD. Dado que la mayoría de clústers también son usados para tareas de Hadoop, Spark está hecho para que YARN también puede gestionar sus recursos. Hemos comentado que Spark tiene una gran adptabilidad en cuanto a arquitectura y de la misma manera lo hace en cuanto a ejecución.

## 2.3 - Lenguaje R

El lenguaje de programación R está basado en el lenguaje S pero con carácter de software libre y orientado a objetos, tiene un enfoque de análisis estadístico y presenta un gran soporte para librerías y paquetes para cargar diferentes funcionalidades para establecer cálculos y gráficos. Además, también tiene compatibilidad con otros lenguajes de programación.

Se trata de uno de los lenguajes más utilizados en investigaciones científicas gracias a las características estadísticas que presenta.

### 2.3.1 - Uso del lenguaje R en Apache Hadoop

Gracias a la flexibilidad de R con su gran soporte para herramientas (en forma de librerías y paquetes) hay una que está diseñada para ejecutar programas en este lenguaje en un clúster de Hadoop. Dicha herramienta se llama "***Rhadoop***" que a través de cuatro paquetes permite a los usuarios manejar y analizar datos en el clúster. Estos paquetes han sido diseñados por la empresa Cloudera y están disponibles en sus versiones CDH3 y CDH4 de máquinas virtuales con herramientas ya preinstaladas de Big Data (como el propio Hadoop).

Cabe destacar que con la última versión de R (3.4.0) y la última versión de Cloudera (CDH5) puede surgir algún que otro problema de compatibilidad al usar esta herramienta. Al hacer algunas pruebas, nos hemos dado cuenta que conectar R con Hadoop es bastante complicado, y que desde RStudio da muchos problemas, así que, finalmente, hemos descartado su uso debido a las grandes complicaciones que acarreaba.

### 2.3.2 - Uso del lenguaje R en Apache Spark

De la misma manera que hay una herramienta para Hadoop y R (aunque sea muy complicado conectarlos) para Spark está la herramienta desde las primeras versiones que se llama ***SparkR*** (que si tenemos Spark instalado nos resultará familiar). Un detalle muy significativo es que esta herramienta solo está presente en versiones muy antiguas de R, y que, en las más actuales, ya no es un paquete a descargar, sino que tenemos que conectarlo manualmente con unos comandos con Spark (lo detallaremos más en el Anexo XV). Hay otros paquetes como ***Sparklyr***, en el que nos centraremos, que te permiten hacer ejemplos para conectar con Spark, incluso con Machine Learning. Trabajaremos con esta última herramienta principalmente ya que en las últimas versiones de R es la que funciona mejor, y comprobaremos todas las principales funcionalidades que nos puede ofrecer.

## 2.4 - ¿Por qué se utiliza Spark?

Desde un inicio Spark fue diseñado para manejar procesado en memoria para que los algoritmos iterativos pudieran desarrollarse sin tener que escribir el resultado tras cada paso por los datos. Esta habilidad es una técnica computacional de alto rendimiento aplicada a análisis de datos avanzados y permite a Spark llegar a velocidades de procesado de hasta 100 veces mayor que otros algoritmos similares que usan MapReduce.

Como se ha comentado con anterioridad Spark es compatible con Scala, Java, Python y R, y tiene un framework integrado para llevar a cabo operaciones de análisis muy avanzadas incluyendo una librería de Machine Learning, un motor de grafos o un motor de análisis en streaming.

Como aliciente, Spark trabaja con cualquier fichero alojado en HDFS o cualquier otro sistema usado por Hadoop, y permite tener ambos entornos instalados a la vez.

Vamos a realizar un vistazo al recorrido de Spark a lo largo de los años:

- 2009 – Nace Spark a manos de Mateiz Zaharia en la UC Berkeley's.
- 2010 – Spark pasa a ser "Open Source", Hadoop está en alza tras su fuerte apuesta por Yahoo!.
- 2013 – El proyecto es donado a la Fundación Apache Software.
- 2014 – La compañía Databricks bate el record de mayor clasificación de datos.
- 2015 – Spark cuenta ya con más de 1000 contribuyentes, convirtiéndolo en uno de los proyectos más activos de la fundación y uno de los mayores proyectos Open Source de Big Data.

Desde entonces se han lanzado 8 versiones estables (la última versión es la 2.2).

Gracias a su fácil implementación y la verstailidad de poder combinarlo con antiguos entornos de Hadoop, Spark se está posicionando como la herramienta Open Source número 1 en cuanto a análisis de Big Data.

# CAPÍTULO 3. ANÁLISIS COMPARATIVO DE TECNOLOGÍAS Y LENGUAJES DE PROGRAMACIÓN

## 3.1 - Uso de Apache Hadoop

En este apartado haremos uso de Apache Hadoop, donde veremos su principal funcionamiento con algunos ejemplos básicos en un entorno donde está totalmente instalado como Cloudera CDH5.

Veremos un ejemplo utilizando la característica principal de Hadoop que es MapReduce, y una serie de ejemplos más allá de esta función con herramientas externas. Dichas herramientas externas forman parte de lo que podemos denominar como ecosistema de Hadoop.

### 3.1.1 – Instalación de Hadoop

Hemos realizado la instalación de Hadoop en Ubuntu 16.04 LTS en modo 'single node' cuya explicación está adjuntada en el Anexo II para realizar unos cuantos ejemplos sencillos.

Por otro lado, hemos utilizado en mayor medida el entorno virtualizado Cloudera CDH5 para utilizar Hadoop, ya que la instalación completa presenta una gran dificultad y no contiene todo el ecosistema, por lo tanto, hemos considerado oportuno no explicarlo y justificarlo en este apartado.

### 3.1.2 - Ejecución y procesado de WordCount

En este apartado del trabajo, veremos como crear un ejemplo con Java en Hadoop, subirlo y procesarlo en el sistema de ficheros HDFS y analizar los tiempos de ejecución para luego poder hacer una comparativa con otras tecnologías y extraer conclusiones.

Utilizaremos tres ficheros de texto (que los colocaremos en la carpeta input) para realizar el ejemplo de WordCount, uno que será un fragmento de un capítulo de Don Quijote de la Mancha, otro que consistirá en una gran parte de un libro de Psicología y el otro una parte del libro de Moby Dick. Evidentemente, el tamaño de los ficheros es distinto, presentando del primero hasta el último una escala gradual de menor a mayor tamaño respectivamente.

El primer paso consiste en crear el programa con las clases Map y Reduce y hacer el .jar para poder ejecutarlo y subirlo mediante comandos. En nuestro caso, hemos utilizado un complemento de Eclipse en el que podíamos crear una clase Map y otra clase para Reduce, para luego llamarlas en la clase main WordCount. El código íntegro (tanto de las clases como el main) se encontrará adjuntado en el Anexo III.

Una vez esté todo correcto, creamos la carpeta input para poder subir los archivos .txt, ejecutaremos el comando para ejecutar el .jar y crearemos la carpeta output para el resultado final, tal y como muestra la Figura 3.1.

```
[cloudera@quickstart hadoop]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera       2447 2017-04-06 21:06 DonQuijote1
[cloudera@quickstart hadoop]$ hadoop fs -ls /user/root
[cloudera@quickstart hadoop]$ hadoop fs -mkdir /user/root/input
[cloudera@quickstart hadoop]$ sudo hadoop jar /home/cloudera/Documents/MarcWordCount.jar hadoop.marc.WordCount /user/root/input /user/root/outpu
t
```

**Figura 3.1** – Comandos para crear la carpeta input y ejecutar el .jar

Una vez ejecutados los comandos anteriores (donde pondremos la ruta donde se encuentra el jar, el nombre del package y las carpetas de entrada y salida), veremos el proceso de MapReduce por consola y el resultado final se encontrará en la carpeta output, donde en el fichero se mostrarán todas las palabras con la cantidad de veces que han aparecido. Un fragmento del proceso de ejecución de MapReduce se muestra en la Figura 3.2.

```
17/04/20 18:41:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1491464986319_0006
17/04/20 18:41:20 INFO impl.YarnClientImpl: Submitted application application_1491464986319_0006
17/04/20 18:41:20 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1491464986319_0006/
17/04/20 18:41:20 INFO mapreduce.Job: Running job: job_1491464986319_0006
17/04/20 18:41:32 INFO mapreduce.Job: Job job_1491464986319_0006 running in uber mode : false
17/04/20 18:41:32 INFO mapreduce.Job:  map 0% reduce 0%
17/04/20 18:41:49 INFO mapreduce.Job:  map 50% reduce 0%
17/04/20 18:41:50 INFO mapreduce.Job:  map 100% reduce 0%
17/04/20 18:41:56 INFO mapreduce.Job:  map 100% reduce 100%
17/04/20 18:41:57 INFO mapreduce.Job: Job job_1491464986319_0006 completed successfully
17/04/20 18:41:57 INFO mapreduce.Job: Counters: 49
```

**Figura 3.2** – Fragmento del proceso de ejecución de MapReduce

Podemos analizar el fichero resultante o ficheros resultantes si te trata de un archivo más grande, que se crea automáticamente al ejecutarse el .jar, vendrá con un nombre por defecto “part-000000”, tal y como podemos ver en la Figura 3.3.

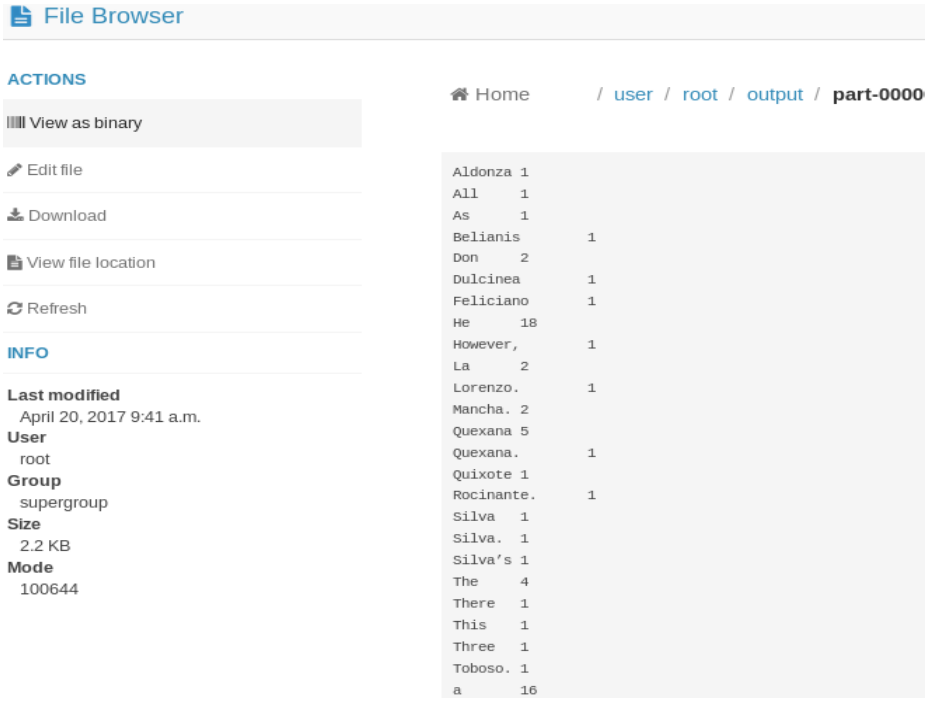



**Figura 3.3** – Fichero resultante del proceso MapReduce contando las palabras del fichero de texto visto desde el Navegador de Cloudera

Si seguimos estos pasos, el WordCount se ejecutará correctamente y sin problemas, y obtendremos los resultados en forma de ficheros de los 3 ficheros iniciales.

Hay varios análisis que podemos efectuar al usar este ejemplo, uno de los más importantes son los tiempos de ejecución y procesado de los ficheros, ya que es una característica importante de Hadoop, haremos una diferenciación entre los ficheros que tenemos. Para ello, miraremos el tamaño de cada archivo, tal y como muestra la Tabla 3.4.

| # Archivo | Nombre del archivo | Tamaño |
|---|---|---|
| 1 | Fragmento de Don Quijote de la Mancha | 35,1 kB |
| 2 | Parte del libro de Psicología | 40 MB |
| 3 | Parte del libro de Moby Dick | 100 MB |

**Tabla 3.4** – Archivos con su correspondiente tamaño

El primero consistirá en el fichero de Don Quijote de la Mancha, que al estar en la carpeta input lo podremos procesar con el .jar. Podemos comprobar el tiempo de ejecución de este primer archivo en la Figura 3.5.

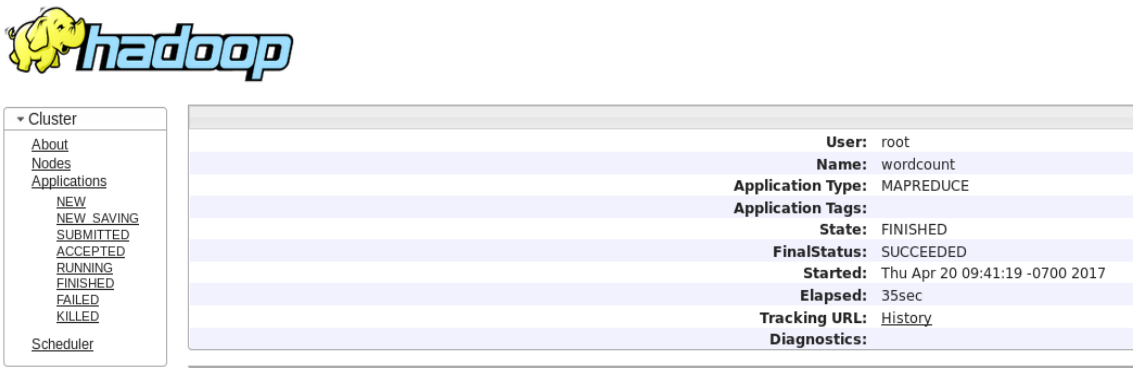



**Figura 3.5** – Características del proceso lanzado con el primer archivo

El primer proceso con dicho fichero tarda ***unos 35 segundos*** en ejecutarse y procesarse, tal y como recogen los datos mostrados en la Figura anterior.

El segundo archivo mucho mayor que el primero que consiste en la gran parte de un libro de Psicología, dónde aparecen muchas más palabras. Repitiendo el mismo procedimiento que el anterior, podemos comprobar el tiempo de ejecución en la Figura 3.6.

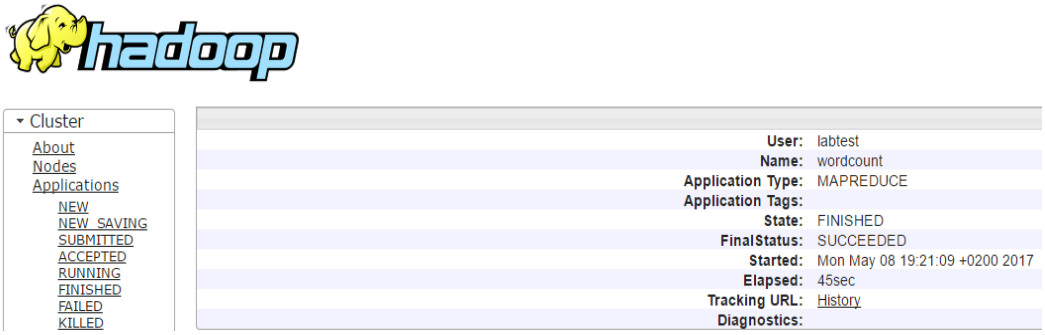



**Figura 3.6** – Características del proceso lanzado con el segundo archivo

Tarda ***unos 45 segundos*** en ejecutarse y procesarse.

El tercer archivo, el mayor de todos, y repitiendo el mismo procedimiento que en los anteriores, podemos comprobar el tiempo de ejecución en la Figura 3.7.

| | |
|---|---|
| **User:** | root |
| **Name:** | wordcount |
| **Application Type:** | MAPREDUCE |
| **Application Tags:** | |
| **State:** | FINISHED |
| **FinalStatus:** | SUCCEEDED |
| **Started:** | Sun Apr 23 04:26:44 -0700 2017 |
| **Elapsed:** | 51sec |
| **Tracking URL:** | History |
| **Diagnostics:** | |

| | |
|---|---|
| **Total Resource Preempted:** | <memory:0, vCores:0> |
| **Total Number of Non-AM Containers Preempted:** | 0 |
| **Total Number of AM Containers Preempted:** | 0 |
| **Resource Preempted from Current Attempt:** | <memory:0, vCores:0> |
| **Number of Non-AM Containers Preempted from Current Attempt:** | 0 |
| **Aggregate Resource Allocation:** | 190840 MB-seconds, 126 vcore-seconds |

**Figura 3.7** – Características del proceso lanzado con el tercer archivo

Tarda ***unos 51 segundos*** en ejecutarse y procesarse.

Después de todos estos resultados, podemos sacar un gráfico general para ver todos los resultados obtenidos y sacar unas conclusiones.

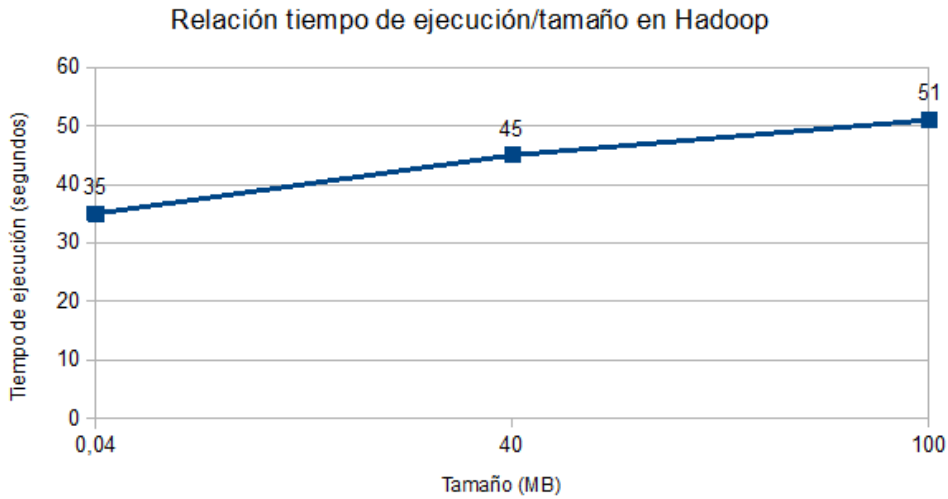



**Gráfico 3.8** – Relación tiempo de ejecución/tamaño de los archivos en Hadoop

Podemos observar que a mayor tamaño del archivo mayor tiempo de procesado evidentemente, pero, por ejemplo, también observamos que para ficheros pequeños el tiempo no es precisamente reducido, por tanto, Hadoop procesa mucho mejor ficheros grandes que pequeños. Además, en este caso hemos utilizado el entorno Cloudera, y el proceso de ejecución y cargado de ficheros se puede hacer tanto por consola como por interfaz gráfica, y realmente no presenta grandes dificultades de uso, en cambio, si lo utilizamos e instalamos por nuestra cuenta, la cosa cambia, ya que la instalación y el uso de Hadoop es bastante complicado.

Además, para crear ejemplos si ya vienen definidos en la suite y podemos modificarlos a través de sus herramientas mucho mejor, sino los tendremos que crear a través de programas como Eclipse o Intellij IDEA los cuales necesitarán aptitudes un poco avanzadas de programación y será mucho más engorroso.

### 3.1.3 – Uso de herramientas externas y ecosistema

Empezaremos con algunos ejemplos con Hadoop, aprovechando la multitud de herramientas externas que se utilizan hoy en día, y que nos ofrece Cloudera y yendo un poco más allá de los ejemplos tradicionales que podemos hacer, ya que Hadoop solo nos ofrece hacer MapReduce.

*Ejemplo 1 – Utilización de bases de datos SQL: Hive e Impala*

Tanto las herramientas Hive e Impala están basadas en bases de datos SQL, por eso las hemos agrupado en el mismo ejemplo, porque tienen sistemas de funcionamiento prácticamente iguales.

Empezaremos con un ejemplo de Hive, en el cual crearemos dos bases de datos, una de usuarios (con los atributos: id, email, language y loc) y la otra de transacciones (con los atributos: id, productId, userId, purchaseAmount y itemDescription). En las Figura 3.9, podemos comprobar la creación de dichas tablas.

```
CREATE EXTERNAL TABLE users(
  id INT,
  email STRING,
  language STRING,
  loc STRING
)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
LOCATION '/user/hive/data/users';
```

```
CREATE EXTERNAL TABLE transactions(
  id INT,
  productId INT,
  userId INT,
  purchaseAmount INT,
  itemDescription STRING
)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
LOCATION '/user/hive/data/transactions';
```

**Figura 3.9** – Creación de las tablas Usuarios y Transacciones

Una vez tengamos las dos tablas creadas y para cada atributo hayamos creado diferentes entradas, haremos la sentencia SQL pertinente para procesarlo con Hadoop y extraer el resultado final.

```
SELECT
  t.productId,
  count(distinct u.loc)
FROM
  transactions t
LEFT OUTER JOIN
  users u on t.userId = u.id
GROUP BY productId;
```

| _c1 |
|---|
| 3 |
| 1 |

✔ Success.

Query History | Saved Queries | Results

| | itemdescription |
|---|---|
| 1 | a jumper |
| 2 | a rubber chicken |

**Figura 3.10** – Sentencia SQL para extraer el resultado combinando las 2 tablas

En dicha sentencia podemos observar como seleccionamos la descripción del objeto y el loc como valor único, y lo selecciona todo de la tabla Transacciones.

Todo seguido, cogemos y realizamos una combinación comenzando con los usuarios de la primera tabla (Usuarios) y luego con el id de usuario de la segunda tabla (Transacciones) y agrupamos los resultados en función de la descripción del objeto.

Una vez realizada dicha sentencia, el resultado será mostrado en pantalla debajo con diferentes pestañas, donde nos aparecerá en Results la solución dada, en este caso con la descripción del objeto y la cantidad de veces que aparece.

Además, al procesar la sentencia, podemos ver el estado de ejecución del proceso en el mismo Hadoop al ejecutarse el MapReduce, por si ha habido algún tipo de problema o ha ido todo bien, en nuestro caso se ha realizado todo correctamente, tal y como muestra la Figura 3.11.


**Figura 3.11** – Procesos de Hive SQL procesados en Hadoop satisfactoriamente

En este mismo apartado, realizaremos también un ejemplo con Impala en el cual tendremos 1 tabla de consumidores, con diferentes atributos cada una. Al hacer la sentencia SQL para extraer el resultado, vamos a seleccionar a los clientes a través de su id, nombre y email, los seleccionaremos también a través de su código ZIP, y finalmente calcularemos la cantidad total por pedido para todos los clientes.

```
1  -- Get email survey opt-in values for all customers
2  SELECT
3    c.id,
4    c.name,
5    c.email_preferences.categories.surveys
6  FROM customers c;
7
8
9
10 -- Select customers for a given shipping ZIP Code
11 SELECT
12   c.id,
13   c.name
14 FROM customers c, c.addresses a
15 WHERE a.key = 'shipping' and a.zip_code = '76710';
16
17
18
19 -- Compute total amount per order for all customers
20 SELECT
21   c.id AS customer_id,
22  c.name AS customer_name,
23   o.order_id,
24   v.total
25 FROM
26   customers c,
27   c.orders o,
28   (SELECT SUM(price * qty) total FROM o.items) v;
```

**Figura 3.12** – Sentencias SQL para extraer el resultado final

Obtenemos el resultado, que nos mostrará el id, nombre y email de todos los clientes de la base de datos, además de la cantidad total por pedido para todos los clientes. En la Figura 3.13 podemos observar dichos resultados.

Query History | Saved Queries | Results

| | id | name | email_preferences.categories.surveys |
|---|---|---|---|
| 1 | 75012 | Dorothy Wilk | false |
| 2 | 17254 | Martin Johnson | true |
| 3 | 12532 | Melvin Garcia | false |
| 4 | 42632 | Raymond S. Vestal | false |
| 5 | 77913 | Betty J. Giambrone | false |
| 6 | 38807 | Rebecca T. Johnson | false |
| 7 | 71843 | David B. Allison | false |
| 8 | 67099 | Jay N. Weaver | false |
| 9 | 83510 | Carol B. Houser | false |
| 10 | 48072 | Octaviana Guiterrez | true |
| 11 | 89830 | Michelle Coffman | false |
| 12 | 98751 | Heather L. Southerland | false |
| 13 | 88702 | Diane Lee | true |

**Figura 3.13** – Resultado final

*Ejemplo 2 – Utilización de Apache Pig*

En esta parte realizaremos un ejemplo con la herramienta Apache Pig, donde a través de un script programado podemos obtener diferentes resultados. En nuestro caso, realizaremos el mismo ejemplo que en el caso de Hive, con la finalidad de ver que, con diferentes procesos y diferentes herramientas, según nos convenga, podemos conseguir los mismos resultados. Utilizaremos el siguiente script (Figura 3.14)

Prova 1

```
USERS = load '/user/hive/data/users' using PigStorage('\t') as (id:int, email:chararray, language:chararray, location:chararray);

TRANSACTIONS = load '/user/hive/data/transactions' using PigStorage('\t') as (id:int, product:int, user:int, purchase_amount:double, description:chararray);

A = JOIN TRANSACTIONS by user LEFT OUTER, USERS by id;

B = GROUP A by product;

C = FOREACH B {
  LOCS = DISTINCT A.location;
  GENERATE group, COUNT(LOCS) as location_count;
};

DUMP C;
```

**Figura 3.14** – Script utilizado para utilizarlo con Apache Pig

Lo primero que realizaremos será cargar las 2 bases de datos anteriormente utilizadas (Usuarios y Transacciones) poniendo la ruta donde se encuentran y cargando todos los atributos asociados. Definimos la variable A, como la combinación de las tablas de Transacciones y Usuarios, y eligiendo de la primera los usuarios y de la segunda los id. La variable B es para ordenar A por producto y agruparla. En la variable C creamos un bucle utilizando la variable B y la variable A, donde buscaremos esta última por ubicación y contaremos cuantas localizaciones hay mediante un grupo.

Una vez este acabado el script y todo correcto, podemos ejecutarlo y compilarlo y ver que no hay ningún problema, tal y como podemos ver en la Figura 3.15


**Figura 3.15** – Compilación correcta del script anterior

El fichero resultante estará presente en el sistema de ficheros HDFS de Hadoop para poder consultarlo y verlo.

*Ejemplo 3 – Utilización de la herramienta MetaStore – Manager*

Hay una utilidad muy interesante dentro del sistema de Cloudera, la cual nos permite ver de forma totalmente gráfica las bases de datos que tenemos creadas, así como todos sus detalles. En nuestro caso, tenemos la de Transacciones y la de Usuarios.

Al usarla, podremos ver todos los detalles de dicha base de datos, como sus propiedades, las columnas y atributos que presenta, con que campos están rellenados cada uno, etc. Es muy útil para revisar todos los datos y para saber si los resultados de cualquier sentencia nos devuelven el resultado correcto.

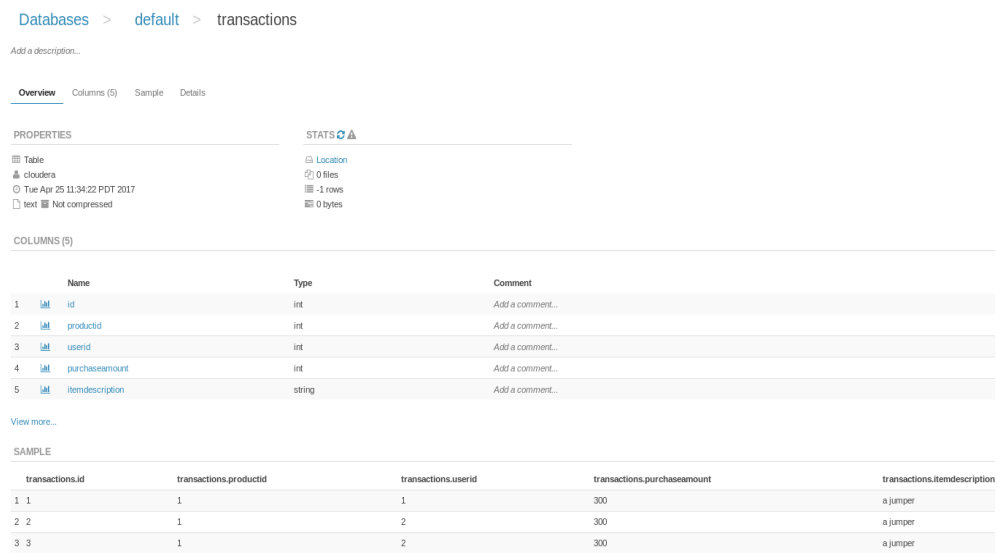



**Figura 3.16** – Detalles de la base de datos de Transacciones

*Ejemplo 4 – Utilización de Solr Search*

Esta herramienta también viene incorporada en Cloudera y se trata básicamente de un motor que potencia la búsqueda y navegación de muchos sitios de Internet, está basado en bibliotecas de Java.

Para probarlo, cogimos un ejemplo de la misma biblioteca de serie que ya venía, e hicimos una búsqueda de Tweets que se habían publicado en 2014 entre distintas franjas horarias y nos mostró diferentes resultados, tal y como muestra el Gráfico 3.17.

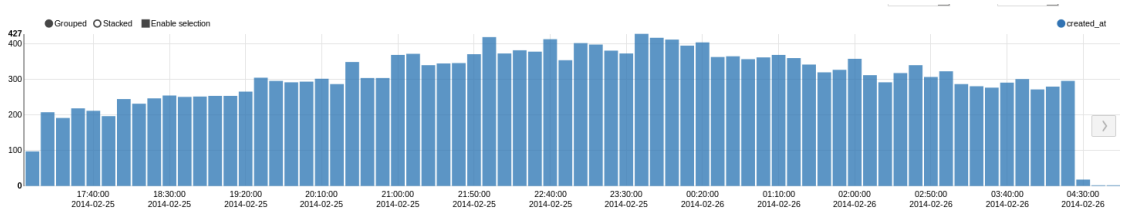



**Gráfico 3.17** – Tweets publicados en 2014, diferenciados por horas

Podemos observar en la parte del gráfico de barras los Tweets lanzados entre diferentes franjas horarias, y a altas horas de la madrugada hay un descenso pronunciado del uso de la red social. Además, podemos ver un diagrama de sectores que nos indica desde que localización en porcentaje se ha lanzado el Tweet, tanto el país como la ciudad exacta.

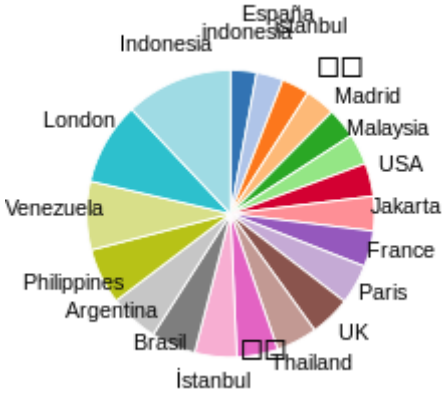



**Gráfico 3.18** – Tweets publicados en 2014, diferenciados por países

También podemos ver algunos ejemplos de los Tweets lanzados por los usuarios en la parte inferior de los resultados.

Es una herramienta útil para procesar datos en el clúster y verlos de forma totalmente gráfica y estadística (siempre que se pueda), para poder realizar un estudio.

*Ejemplo 5 – Utilización de JobDesigner y Apache Oozie*

A parte de poder ejecutar los programas mediante comandos de Hadoop por consola, también se nos ofrece una alternativa para hacerlo gráficamente para los que no estemos familiarizados con el entorno, dicha alternativa es JobDesigner. Esta herramienta nos facilita la compilación de programas creados por nosotros, añadiendo unos campos para adjuntar el fichero .jar resultante y algunas propiedades para poder compilarlo, tal y como muestra la Figura 3.19.

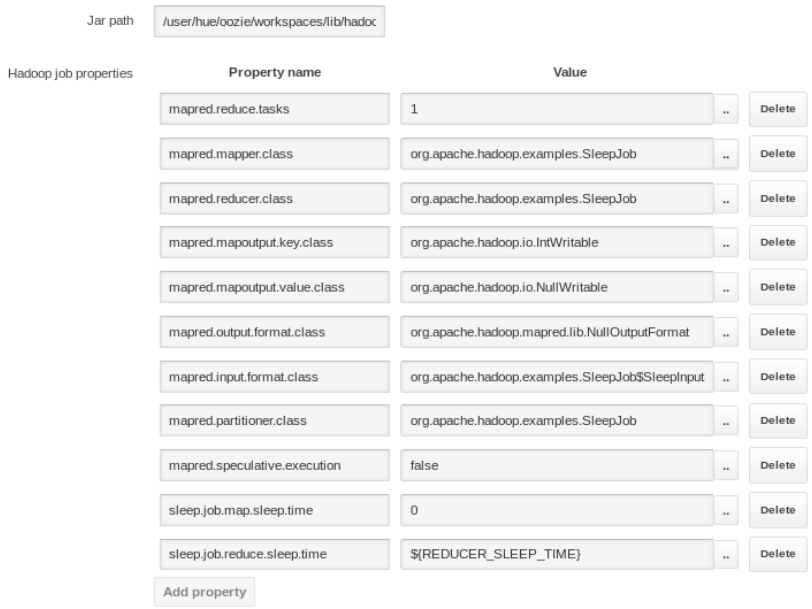



**Figura 3.19 –** Utilización de JobDesigner para añadir el .jar y las propiedades de Hadoop

En este ejemplo utilizamos un WordCount creado por nosotros mismos y haciendo la comparación entre la ejecución de JobDesigner y el resultado final por consola y haciendo el .jar desde Eclipse, evidentemente el resultado será el mismo.

Una vez tenemos el .jar cargado y todo correcto, lo ejecutaremos mediante Apache Oozie con tal de ejecutar el proceso de MapReduce de Hadoop.
Apache Oozie es un sistema de programación de flujo de trabajo basado en servidor para gestionar los trabajos de Hadoop. Los flujos de trabajo están definidos como una colección de nodos de flujo de control y de acción. Los de control definen el principio, el final de un flujo de trabajo y la ruta de ejecución. Los de acción son los flujos que activan la tarea asociada. En nuestro caso, lo utilizaremos para ejecutar el MapReduce.

Al ejecutarlo, vemos que el proceso de MapReduce se establece correctamente y nos deja el resultado en el sistema de ficheros HDFS. Al trabajar con el flujo de trabajo se hace todo de forma automática siempre y cuando pongamos todos los parámetros de la forma correcta.

Estos son los principales ejemplos que hemos podido probar en el entorno de Cloudera, hay más evidentemente, pero aquí solo hemos mostrado unos cuantos, ya que otros no acababan de funcionar del todo bien y no podíamos

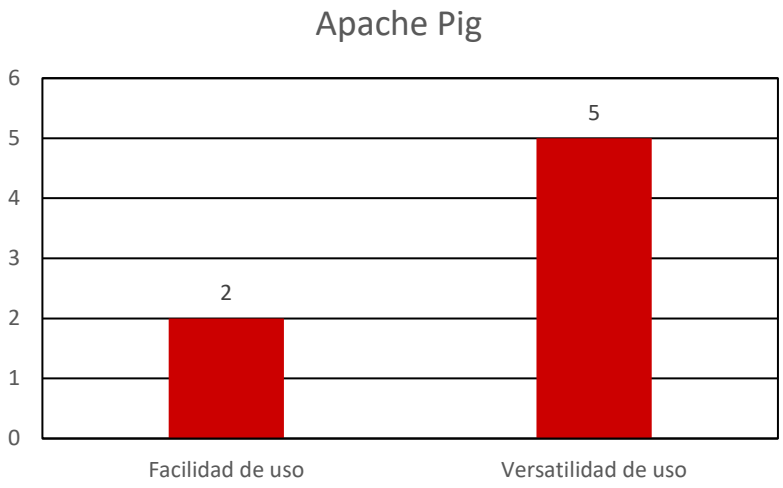
mostrarlos adecuadamente, así que hemos mostrado unos cuantos que también resultaban interesantes.

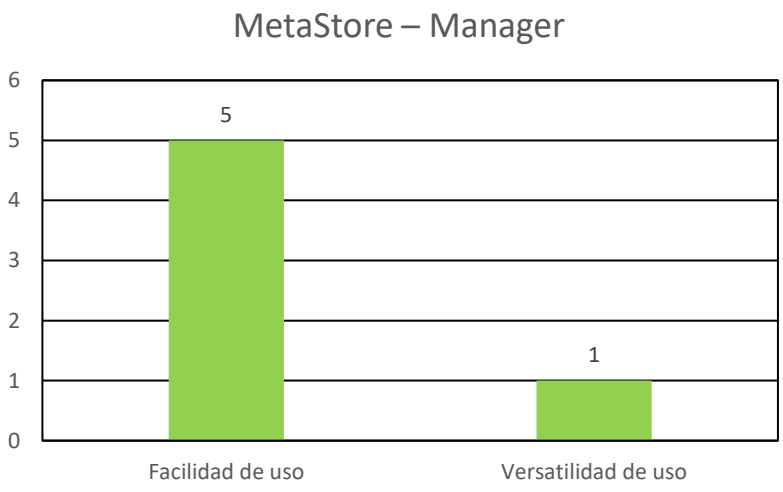
### 3.1.4 – Conclusiones de Hadoop

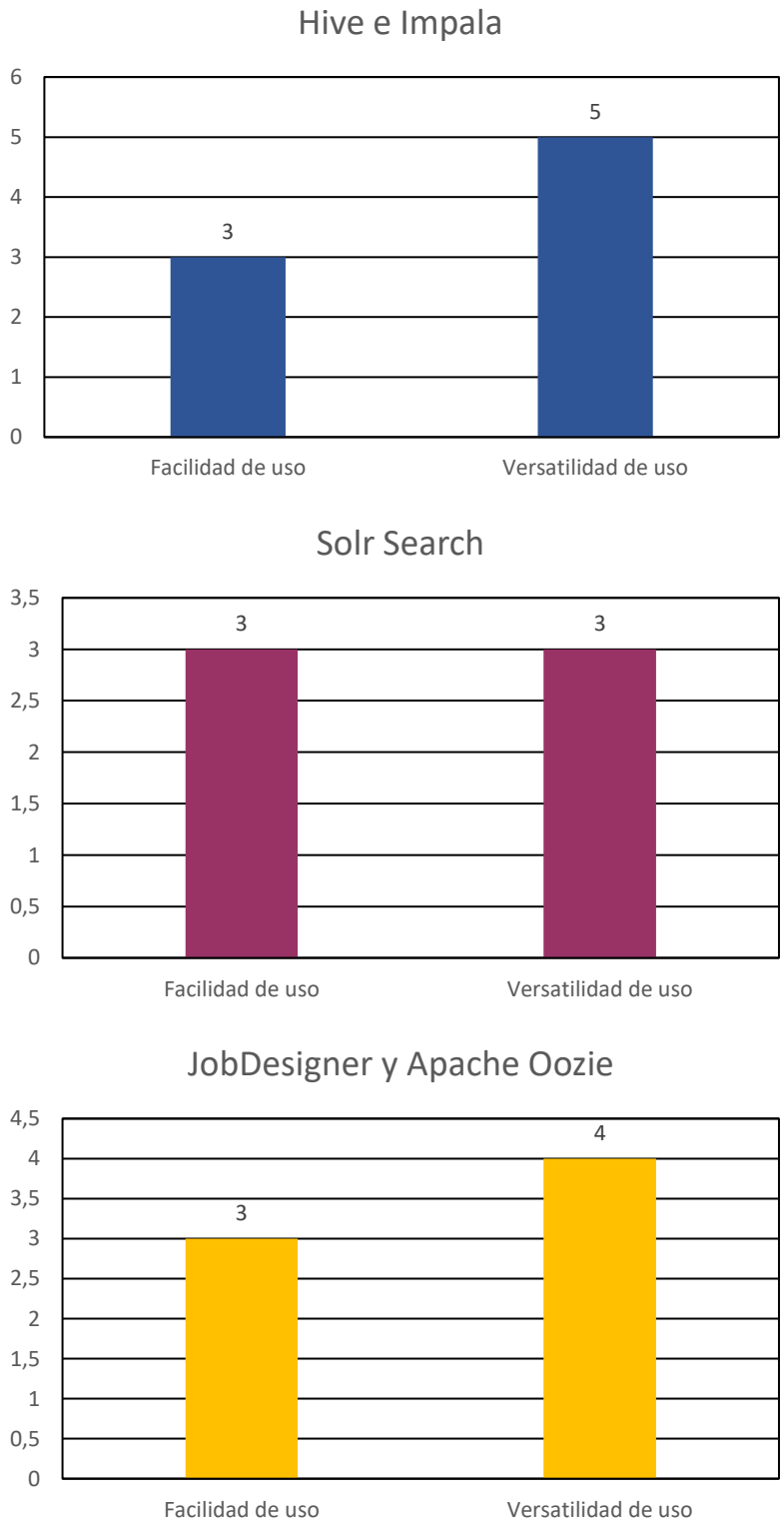
En esta parte, extraeremos unas primeras conclusiones de las herramientas externas de Hadoop, ya que las conclusiones finales se encuentran en la parte final del capítulo. Por otro lado, destacamos que estas conclusiones serán un poco diferentes a las de Spark, ya que no son librerías y, por lo tanto, las características y los métodos de evaluación son diferentes.

A lo largo del ciclo de vida de Hadoop se han ido añadiendo multitud de herramientas externas que van más allá de la función MapReduce, aquí solo realizaremos un análisis de las anteriormente vistas. Para evaluarlas, nos centraremos en diferentes características, entre las cuales se encuentran la facilidad de uso (es decir, la complicación al utilizar la herramienta en sí) y la versatilidad de que tiene para usarse en diferentes programas o con otras herramientas, y en una escala del 0 al 5 las puntuaremos, siendo la nota de 5 la mejor puntuación.

A continuación, mostramos las gráficas de evaluación de cada herramienta tal y como hemos comentado anteriormente:






**Gráficos 3.20** – Evaluación de herramientas externas, dividido en 'Facilidad de uso' y 'Versatilidad de uso'

Si analizamos un poco por encima los resultados mostrados en los gráficos anteriores y destacando siempre nuestra consideración y recomendación personal, podemos observar que en cuanto a versatilidad de uso, empatan en el primer puesto las herramientas Hive, Impala y Apache Pig, debido a que las dos primeras son para realizar sentencias de bases de datos en SQL y se pueden

utilizar en muchísimos ámbitos (sin necesidad de que sean de Big Data) y la última se basa en scripts programados para ejecutar diferentes funciones que también se pueden utilizar en cualquier ámbito. Por otro lado, la facilidad de uso entre estas dos, personalmente creemos que es mayor en Hive e Impala debido al lenguaje SQL.

La herramienta MetaStore Manager presenta la mayor facilidad de uso de todas las herramientas, gracias en gran medida a su muy clara interfaz web y visual que nos permite observar todos y cada uno de los atributos de nuestra base de datos, por contra, solo hemos podido comprobar esta utilidad relacionada con bases de datos que hace que la versatilidad de uso sea muy limitada.

Por otro lado, la herramienta Solr Search no presenta grandes complicaciones de uso para analizar diferentes datos de la biblioteca, en cambio, la versatilidad se ve un poco mermada ya que no hemos visto muchas aplicaciones exteriores fuera de Hadoop.

Finalmente, las herramientas JobDesigner y Apache Oozie, creemos que sirven sobre todo por si no estamos muy familiarizados con los comandos de Hadoop, para facilitar la subida de archivos .jar y su posterior ejecución, son de gran ayuda y con una facilidad de uso media. La versatilidad es grande dentro del ámbito de Big Data, ya que con cualquier programa podemos hacer uso de ellas, pero en cambio, fuera de ese ámbito no vemos muchas más aplicaciones.

## 3.2 - Uso de Apache Spark

En esta parte del capítulo veremos Spark y el lenguaje con el que está diseñado, Scala, lo cual esperamos que nos facilite su uso, así como la información disponible.

### 3.2.1 - Instalación de Spark

Al inicio del trabajo la última versión de Spark era la 2.1.0 que será la que utilizaremos a lo largo de todas nuestras pruebas. Spark como se ha comentado, es joven y por lo tanto recibe actualizaciones muy a menudo, el hecho de tener un equipo activo detrás se nota mucho en la documentación disponible.

La instalación es más sencilla que la de Hadoop, solo hace falta descargar el archivo .tgz e instalarlo en nuestro sistema (en el Anexo XII detallaremos los pasos de instalación) Spark viene con algunos ejemplos integrados, en la Figura 3.21 podemos observar cuatro líneas en las que leemos el número de palabras en el Readme de Spark mismo.

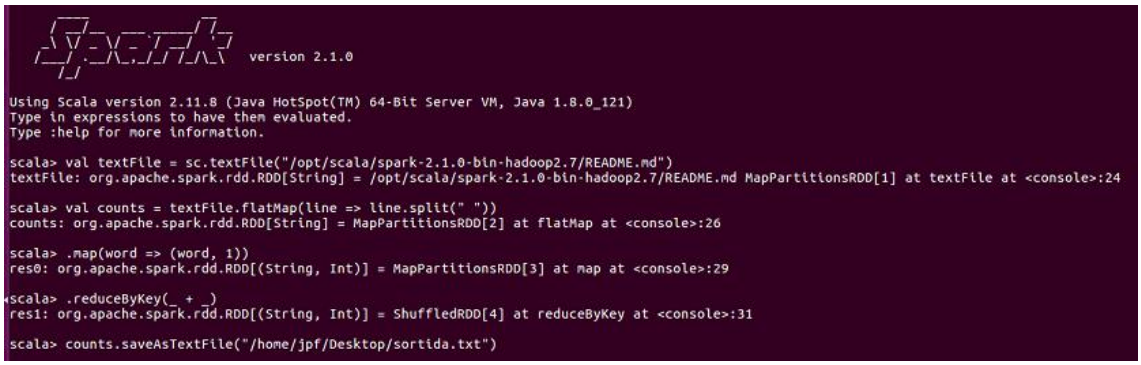



**Figura 3.21** – Consola de Spark (Spark Shell)

La consola de Spark ofrece una manera fácil de probar algunas líneas de código, pero para implementar todo un programa no será suficiente, nos hace falta un IDE en el que programar con Scala. Nosotros nos hemos decantado por IntelliJ IDEA (referido como IDEA a partir de ahora) porque estamos familiarizados con él, otra opción muy común por su integración con Maven puede ser Eclipse.

La instalación de IDEA, como la de Spark, no es difícil, los problemas vienen cuando queremos integrar ambos. Para empezar, queremos trabajar con Scala, pero IDEA nos instalará por defecto la última versión de Scala, la cual puede dar problemas de clases inexistentes en Spark ya que este usa la versión 2.11.8.

### 3.2.2 - Ejecución y procesado de WordCount

Tras la instalación de Spark e IDEA en nuestro sistema necesitaremos poder "pasar" nuestro código del uno al otro para poder probarlo, para hacerlo instalaremos la herramienta SBT que, además de otras cosas, nos permitirá crear un archivo .jar con todas las dependencias y así poder ejecutarlo desde consola con el comando *spark-submit*.

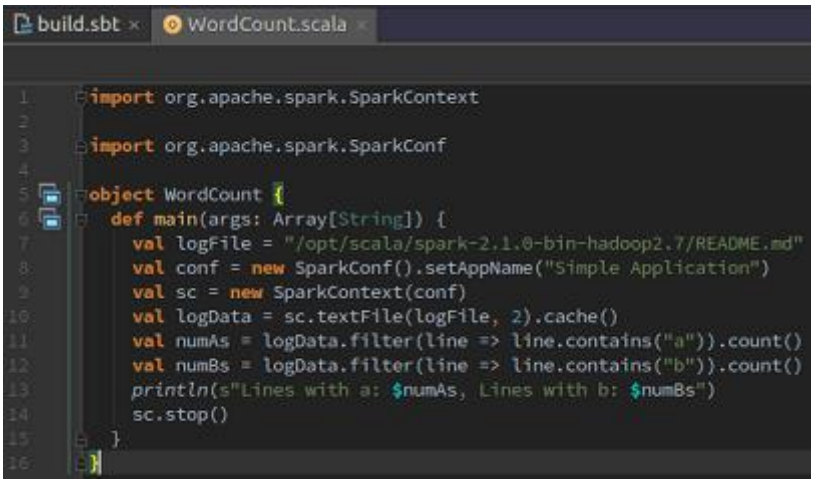



**Figura 3.22** - Pequeño código escrito en Scala en el IDE IDEA

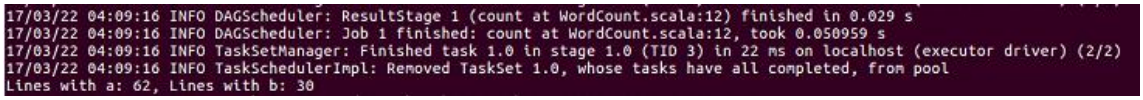



**Figura 3.23** - Resultado tras ejecutar archivo .jar

Como vemos en las Figuras 3.22 y 3.23 el código usado es muy básico, simplemente cuenta el número de “a” y “b” en el texto.

Para poder hacer comparaciones con otras tecnologías vamos a ejecutar un WordCount (que cuente palabras realmente) sobre archivos de diferente tamaño para ver las velocidades de procesado.

De igual manera que hemos hecho con Hadoop, vamos a procesar tres archivos de texto, un fragmento de Don Quijote, el libro de Moby Dick y otro llamado “El despertar del hombre”. Como podemos ver en la Figura 3.24 el programa guarda el resultado en una carpeta “output” y no devuelve nada por pantalla.

```
17/04/24 23:22:16 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1). 2080 bytes result sent to driver
17/04/24 23:22:16 INFO scheduler.DAGScheduler: ResultStage 1 (saveAsTextFile at WordCount.scala:26) finished in 1.194 s
17/04/24 23:22:16 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 1197 ms on localhost (1/1)
17/04/24 23:22:16 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
17/04/24 23:22:16 INFO scheduler.DAGScheduler: Job 0 finished: saveAsTextFile at WordCount.scala:26, took 3.388268 s
17/04/24 23:22:16 INFO spark.SparkContext: Invoking stop() from shutdown hook
```

**Figura 3.24** - WordCount aplicado al libro de Psicología

Vemos que el tiempo que tarda en procesar es bastante bajo, en este caso para un texto de 40 MB ha tardado poco ***más de 3 segundos***, vamos a ver la comparación con los otros textos en el Gráfico 3.25.

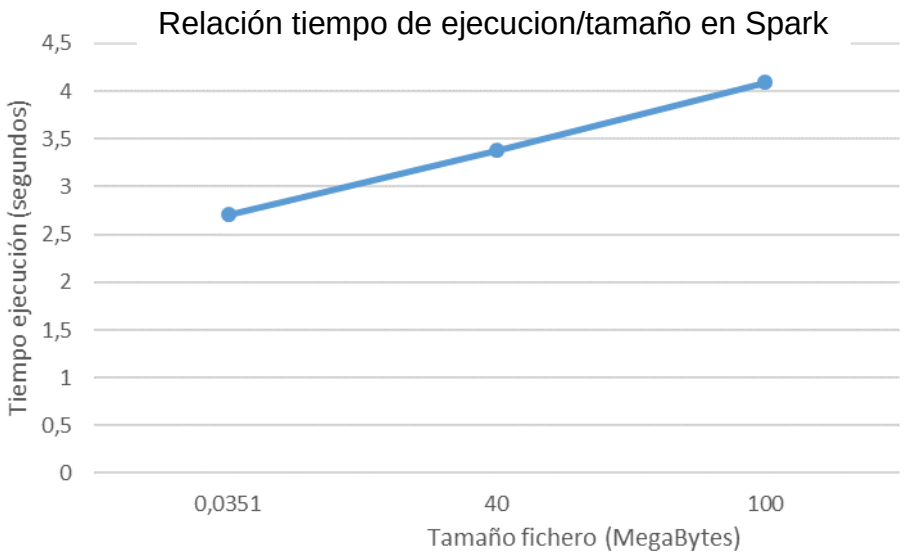



**Gráfico 3.25 -** Relación tiempo de ejecución/tamaño de los archivos en Spark

Es difícil interpolar la recta con tan solo 3 puntos, podría ser lineal o podría ser logarítmica. Lo que si podemos extraer es que los tiempos son muy prometedores y más si tenemos en cuenta que estas pruebas se realizan en modo Standalone, la ventaja de una tecnología como Spark, es que permite trabajar con datos distribuidos de manera muy rápida.

## 3.2.2 – Librerías

Como ya se ha comentado, Spark en contrapartida a Hadoop, lleva una serie de librerías ya incluidas (como SQL, DataFrames, Machine Learning, Streaming, etc.) que pueden ser usadas simultáneamente y que también pueden ser nombradas como ecosistema de Spark. Además, su rápida extensión entre empresas y organizaciones ha hecho que su comunidad crezca muy rápido, haciendo que haya disponible también un número considerable de librerías externas.

### *3.2.2.1 - Spark SQL/GraphX*

Primero vamos a ver en conjunto dos librerías, Spark SQL y GraphX. Spark SQL permite la consulta de datos usando SQL u otra API de DataFrame.
GraphX, por otro lado, es la API para gráficos y cálculo gráfico paralelo, unifica ETL, análisis exploratorios y computación gráfica iterativa en un solo sistema.
Spark realiza operaciones sobre un RDD que da como resultado otro RDD, gráficamente esto se puede entender como un gráfico directo acíclico, la transformación se realiza en una sola dirección (directo) y no se puede volver atrás (acíclico). Hay que tener en cuenta también el concepto de Nodo que son los puntos en el gráfico, típicamente objetos, personas, lugares… y vértice, que son las relaciones entre los nodos (Figura 3.26).

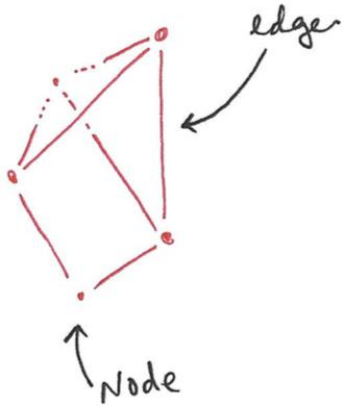



**Figura 3.26** - Esquema de una gráfica

*Ejemplo 1 – Uso de GraphX y Spark SQL*

El código completo de este ejemplo está disponible en el Anexo VII. Este pequeño programa extrae los datos de un fichero que contiene información sobre aeropuertos, que vuelos salen de que aeropuerto, donde van, si ha habido incidencias, etc.

Como podemos ver en la Figura 3.27, tras procesar el fichero de texto crea las variables que contendrán nuestros RDD, mapea y define los nodos y los vértices que conformará nuestro grafo.

```
// parse the RDD of csv lines into an RDD of flight classes
val flightsRDD = textRDD.map(parseFlight).cache()

// create airports RDD with ID and Name
val airports = flightsRDD.map(flight => (flight.org_id, flight.origin)).distinct
airports.take(1)

// Defining a default vertex called nowhere
val nowhere = "nowhere"

// create routes RDD with srcid, destid, distance
val routes = flightsRDD.map(flight => ((flight.org_id, flight.dest_id), flight.dist)).distinct
```

**Figura 3.27** - Parte del código del ejemplo de GraphX

Finalmente, con comandos SQL podemos extraer la información que deseamos de nuestra gráfica.

Como vemos GraphX es una herramienta muy útil cuando tenemos datos que están correlados. Las gráficas no son eficientes para ser procesadas por ordenador, pero si son muy intuitivas para nuestro entendimiento, gracias a GraphX podemos mantener el coste computacional bajo ofreciéndonos una manera fácil de entender los datos que procesamos. Google, en sus inicios, usaba el método “PageRank” para dar importancia a las páginas web que procesaba y así devolver el resultado en un orden determinado, el comando que usamos para obtener el aeropuerto más “importante” es dicho “PageRank”

### *3.2.2.2 - Spark Streaming*

Esta librería brinda a Spark con una API que permite escribir código para Streaming como lo harías con cualquier otro trabajo por lotes. Con esta funcionalidad, podemos hacer un seguimiento de las estadísticas de una página en tiempo real, entrenar un modelo de Machine Learning, auto detectar anomalías, etc.

*Ejemplo 2 – Uso de Spark Streaming*

El código completo de esta aplicación se puede encontrar en el Anexo VIII.

El código que ejecutaremos estará monitorizando un Chat Online. Usaremos una página web en la que podemos crear nuestras propias salas de chat (slack.com) y desde cuya API podamos obtener un Token para hacer el seguimiento en tiempo real.

```
object SlackStreamingApp {

  def main(args: Array[String]) {
    val conf = new SparkConf().setMaster(args(0)).setAppName("SlackStreaming")
    val ssc = new StreamingContext(conf, Seconds(5))
    val stream = ssc.receiverStream(new SlackReceiver(args(1)))
    stream.print()
    if (args.length > 2) {
      stream.saveAsTextFiles(args(2))
    }
    ssc.start()
    ssc.awaitTermination()
  }

}
```

**Figura 3.28** – Código principal “Main” de nuestro ejemplo

En la Figura 3.28 vemos como de sencillo es el main de nuestro ejemplo, simplemente crea un StreamingContext y llama a la función receiverStream cada 5 segundos que guardará la información obtenida. Spark Streaming se basa en "discretized Streams" por lo que a cada marca temporal nos llegará una secuencia de RDDs.

En la Figura 3.29 veremos que la clase SlackReceiver a la que se llama desde el main contiene el código para conectarse al chat que hemos creado y las funciones para monitorizar la página y dejar de hacerlo.

```
class SlackReceiver(token: String) extends Receiver[String](StorageLevel.MEMORY_ONLY) with Runnable with Logging {

  private val slackUrl = "https://slack.com/api/rtm.start"

  @transient
  private var thread: Thread = _

  override def onStart(): Unit = {...}

  override def onStop(): Unit = {...}

  override def run(): Unit = {...}

  private def receive(): Unit = {...}

  private def webSocketUrl(): String = {
    val response = Http(slackUrl).param("token", token).asString.body
    JSON.parseFull(response).get.asInstanceOf[Map[String, Any]].get("url").get.toString
  }

}
```

**Figura 3.29** - Classe SlackReceiver

Como podemos ver mirando el código este programa, guardará cada 5 segundos una carpeta con la información que haya en el chat. En la Figura 3.30 podemos ver diversas carpetas correspondientes a cada "timestamp" y en ellas habrá textos como el que se muestra.

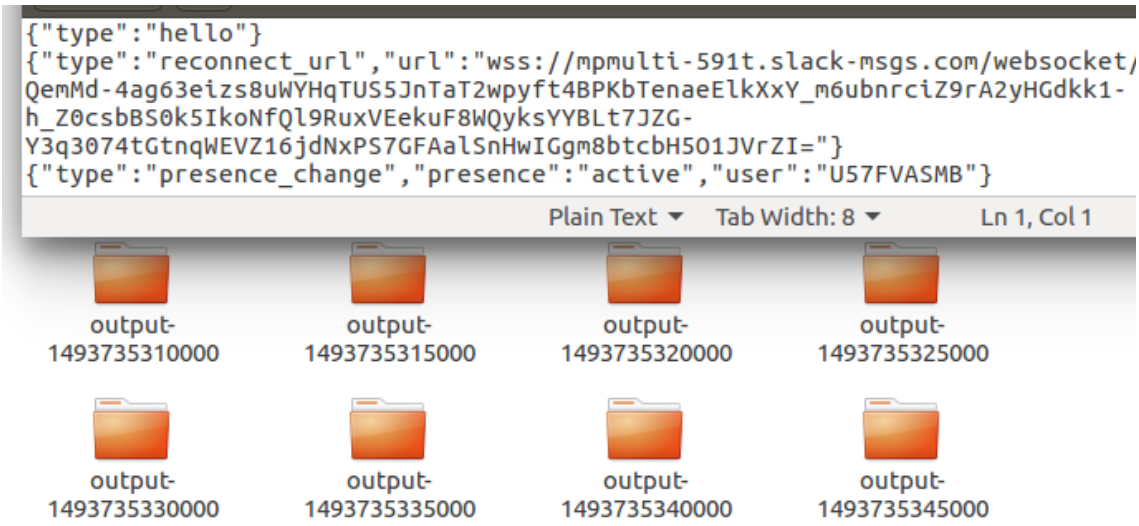



**Figura 3.30** - Carpetas de los outputs resultantes tras ejecutar el programa

Como se ve este ejemplo de Spark Streaming es un programa básico, pero eficaz y de monitorización, no es difícil de escribir y sobre esta base el usuario es libre de añadir más librerías o poner las funcionalidades que desee.

*3.2.2.3 - MLlib*

A medida que se crean productos y servicios de datos centrados en el usuario más diversos hay una necesidad creciente de Machine Learning, que puede ser usado precisamente para desarrollar personalizaciones, recomendaciones o predicciones. La ventaja que nos aporta la librería de Machine Learning de Spark (MLlib) es que podemos centrarnos en los problemas y modelos dejando a un lado la complejidad que conllevan los datos distribuidos.

*Ejemplo 3 - Uso de MLlib*

En el último ejemplo de la parte de Spark hemos querido enfatizar la parte de Big Data con un ejemplo que fácilmente podríamos estar usando diariamente, se trata de un recomendador de películas.

Uno de los algoritmos más usados para recomendadores es el filtro colaborativo. Esta técnica llena las entradas vacías de una matriz de elementos-usuarios. MLlib utiliza ALS (Alternating Least Squares) para aprender una serie de factores latentes que describen a los usuarios y a los productos.

Usaremos un dataset cortesía de MovieLens, que contiene 10 millones de puntuaciones de 72000 usuarios sobre 10000 películas. El ejemplo empieza con un pequeño script de Python en el que daremos puntuación a unas películas del 0 al 5.

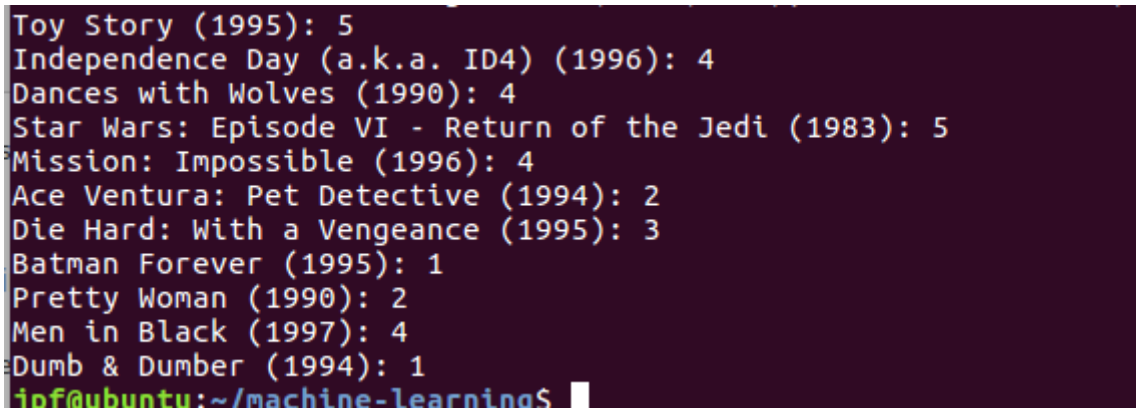



**Figura 3.31 -** Ejemplo de “rating” personal

Dado que el código es más largo que en ejemplos anteriores no añadiremos ninguna Figura al respecto, en el Anexo IX se puede encontrar el código utilizado para usar en Spark.

El programa lo que hace es dividir nuestro dataset en 3 (entrenamiento, validación y test), entrena diferentes modelos (con diferentes parámetros) y escoge el mejor. Finalmente, devuelve un listado de recomendaciones y la mejora del modelo escogido respecto a un modelo base (sin ningún tipo de entrenamiento).


**Figura 3.32** - Recomendaciones en base al mejor modelo

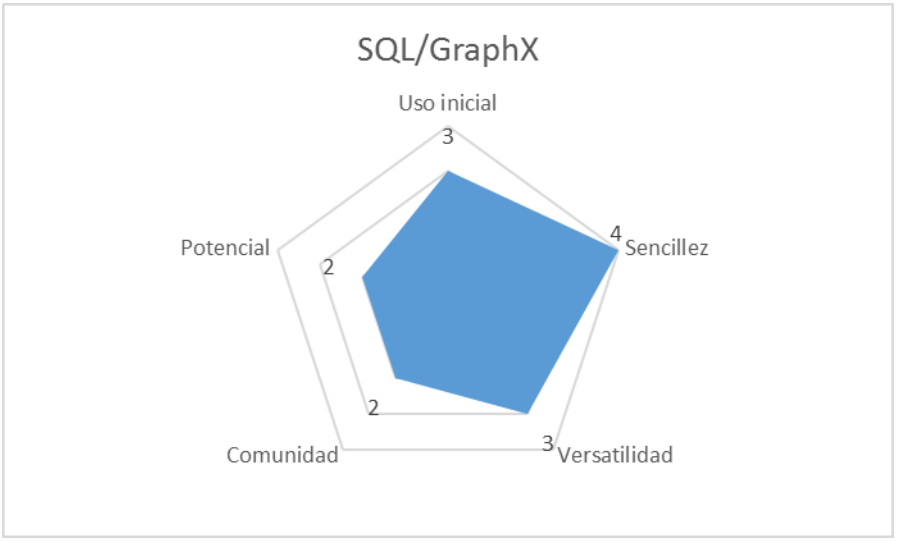
Como vemos, la mejora respecto a un modelo base es de un 22%, que no está nada mal considerando que el recomendador ha tardado alrededor de 30 segundos en entrenarse, escoger el mejor modelo y hacer las recomendaciones en base a 10 puntuaciones.

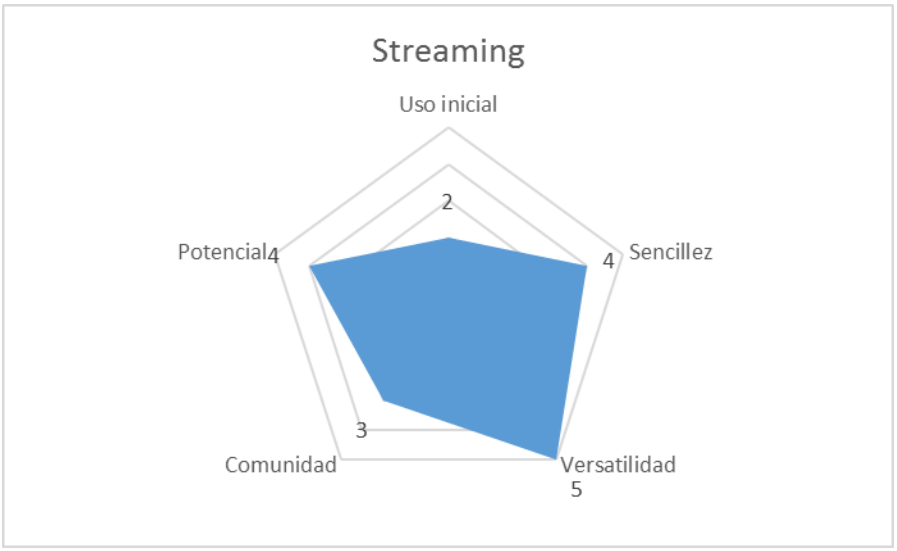
Como nota personal, no son las mejores recomendaciones que me podrían hacer, pero de nuevo, por la complejidad del programa, no está nada mal. Anteriores resultados han sido algo mejores.

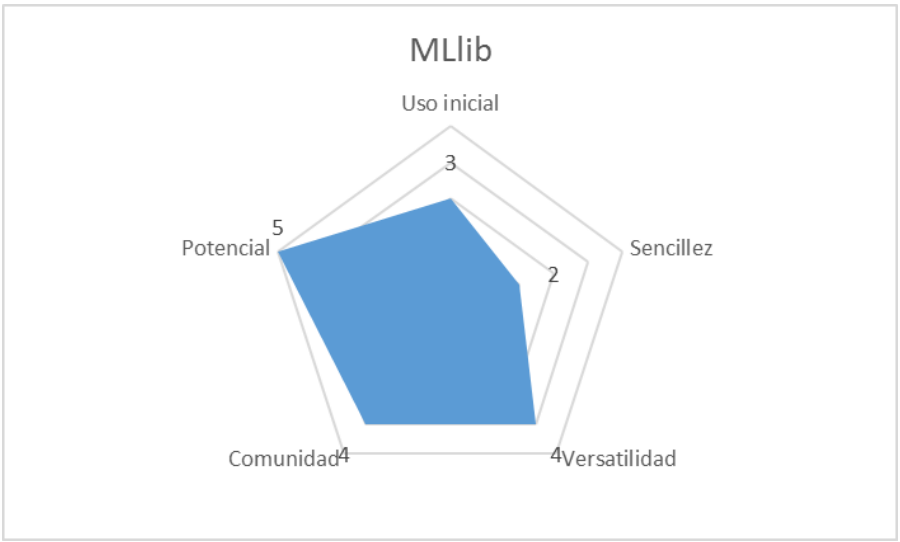
### 3.2.3 – Conclusiones de Spark

Las librerías que hemos visto son algunas de las que vienen integradas y pese a la gran comunidad que hay, el número de aportaciones de terceros, aunque considerable, no es impresionante. Entre ellas destacan spark Azure que, como su nombre indica, permite conectar Spark con Microsoft Azure, o Thunder que permite analizar datos neuronales a gran escala.

Los gráficos siguientes son una representación de las conclusiones que extraemos tras el uso de algunas de las librerías:




**Gráficos 3.33** - Comparativa entre librerías.

Todas las categorías están puntuadas del 1 al 5, siendo 5 el caso más favorable.

En primer lugar, uso inicial, esto indica si es fácil o no empezar a usar la librería con la información que nos da la página de Spark. Sencillez, está ligado al uso inicial e indica la facilidad de entender y escribir código por parte del usuario. Versatilidad hace referencia a la facilidad de adaptación con otras librerías o con otras funcionalidades de Spark. Con comunidad nos referimos a la información extraoficial, así como ejemplos y tutoriales que podemos encontrar en Internet. Finalmente, potencial es el indicador más subjetivo y resume si creemos que la librería aporta un valor añadido.

Como vemos cada una de las tres librerías destaca en apartados diferentes, siendo la peor (a nivel global) GraphX. Pese a que esta tiene un uso muy sencillo, si conocemos comandos SQL, nos resultará más fácil y que el hecho de hacer asociaciones intramatriciales tiene muchas posibilidades, no parece que GraphX tenga mucho éxito; existen otras herramientas más intuitivas y más específicas para hacer el trabajo que hace esta librería.
Un escenario con Spark Streaming es extremadamente sencillo, no gracias a las explicaciones oficiales, pero si a otros tutoriales. Además, su implementación junto a otras librerías hace que sea una herramienta de análisis extremadamente interesante.

Por último, MLlib pese a no ser demasiado sencilla, ya que hay que entender el procesado de datos y diferentes algoritmos, es una librería que ofrece muchas posibilidades, es fácil juntarla con otras librerías (como Spark Streaming) y la mayor parte de la comunidad está involucrada con este tema.

## 3.3 - Uso de Apache Spark con lenguaje R

En esta parte del capítulo, nos centraremos en el uso de Apache Spark junto con el lenguaje R, y enseñaremos las características principales para utilizarlo junto con elementos de Big Data.

### 3.3.1 - Uso de la librería dplyr

***dplyr*** es una librería de funciones para analizar y manipular datos: dividir grandes colecciones de datos, aplicar una función a cada parte y reagruparlas, y también aplicar filtros, ordenar y juntar datos.

*Ejemplo 1 – R con Spark – Vuelos con retraso:*

Explicación del código general:

Analizando la parte del código adjunta expuesta, empezaremos instalando el paquete "*sparklyr*" que nos llevará unos minutos, lo cargaremos con "*library(sparklyr)*" para poder utilizarlo e instalaremos la versión de Spark que más nos interese (recomendamos las últimas versiones). La conexión con Spark se puede realizar de forma local como contra un clúster remoto, en nuestro caso nos conectaremos localmente a Spark mediante la función "*spark_connect*". La

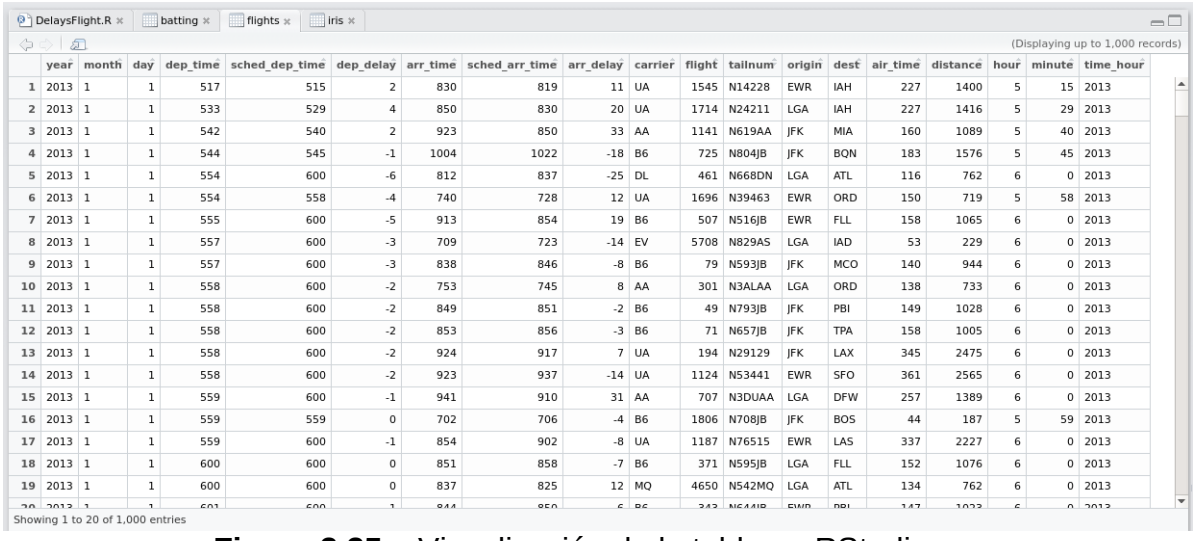
función "*sc*" nos proporciona una conexión remota a los datos que se encuentren en Spark.

```
#Instalamos el paquete "sparklyr"
install.packages("sparklyr")
#Utilizamos el paquete previamente instalado (sparklyr)
library(sparklyr)
#Instalamos, en nuestro caso, la versión en local 2.0.0 de Spark para utilizarla en RStudio (la versión que
tenemos fuera del programa, es la 2.1.0)
spark_install(version = "2.0.0.")
#Iniciamos y hacemos la conexión remota con Spark
sc <- spark_connect(master = "local")
```

**Figura 3.34** – Fragmento de la parte inicial del código

La conexión local de Spark en RStudio no propicia ningún tipo de problema ni complicación (sobre todo si se trata de la última versión). Gracias a ello, podremos aprovecharnos para ejecutar un código utilizando funciones y procesos de Spark.

En este primer ejemplo, representaremos los vuelos con retrasos de la ciudad de Nueva York. Para ello utilizaremos el paquete "*dplyr*" que previamente tendremos que instalar, mediante este paquete copiaremos los datos de R a Spark mediante la función "*copy_to*". Estos datos se copiarán en forma de tabla en Spark para que podamos manipularlos mejor y podremos observar dichas tablas en el propio RStudio, tal y como muestra la siguiente Figura 3.35.


DelaysFlight.R × batting × flights × iris ×

(Displaying up to 1,000 records)

| | year | month | day | dep_time | sched_dep_time | dep_delay | arr_time | sched_arr_time | arr_delay | carrier | flight | tailnum | origin | dest | air_time | distance | hour | minute | time_hour |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2013 | 1 | 1 | 517 | 515 | 2 | 830 | 819 | 11 | UA | 1545 | N14228 | EWR | IAH | 227 | 1400 | 5 | 15 | 2013 |
| 2 | 2013 | 1 | 1 | 533 | 529 | 4 | 850 | 830 | 20 | UA | 1714 | N24211 | LGA | IAH | 227 | 1416 | 5 | 29 | 2013 |
| 3 | 2013 | 1 | 1 | 542 | 540 | 2 | 923 | 850 | 33 | AA | 1141 | N619AA | JFK | MIA | 160 | 1089 | 5 | 40 | 2013 |
| 4 | 2013 | 1 | 1 | 544 | 545 | -1 | 1004 | 1022 | -18 | B6 | 725 | N804JB | JFK | BQN | 183 | 1576 | 5 | 45 | 2013 |
| 5 | 2013 | 1 | 1 | 554 | 600 | -6 | 812 | 837 | -25 | DL | 461 | N668DN | LGA | ATL | 116 | 762 | 6 | 0 | 2013 |
| 6 | 2013 | 1 | 1 | 554 | 558 | -4 | 740 | 728 | 12 | UA | 1696 | N39463 | EWR | ORD | 150 | 719 | 5 | 58 | 2013 |
| 7 | 2013 | 1 | 1 | 555 | 600 | -5 | 913 | 854 | 19 | B6 | 507 | N516JB | EWR | FLL | 158 | 1065 | 6 | 0 | 2013 |
| 8 | 2013 | 1 | 1 | 557 | 600 | -3 | 709 | 723 | -14 | EV | 5708 | N829AS | LGA | IAD | 53 | 229 | 6 | 0 | 2013 |
| 9 | 2013 | 1 | 1 | 557 | 600 | -3 | 838 | 846 | -8 | B6 | 79 | N593JB | JFK | MCO | 140 | 944 | 6 | 0 | 2013 |
| 10 | 2013 | 1 | 1 | 558 | 600 | -2 | 753 | 745 | 8 | AA | 301 | N3ALAA | LGA | ORD | 138 | 733 | 6 | 0 | 2013 |
| 11 | 2013 | 1 | 1 | 558 | 600 | -2 | 849 | 851 | -2 | B6 | 49 | N793JB | JFK | PBI | 149 | 1028 | 6 | 0 | 2013 |
| 12 | 2013 | 1 | 1 | 558 | 600 | -2 | 853 | 856 | -3 | B6 | 71 | N657JB | JFK | TPA | 158 | 1005 | 6 | 0 | 2013 |
| 13 | 2013 | 1 | 1 | 558 | 600 | -2 | 924 | 917 | 7 | UA | 194 | N29129 | JFK | LAX | 345 | 2475 | 6 | 0 | 2013 |
| 14 | 2013 | 1 | 1 | 558 | 600 | -2 | 923 | 937 | -14 | UA | 1124 | N53441 | EWR | SFO | 361 | 2565 | 6 | 0 | 2013 |
| 15 | 2013 | 1 | 1 | 559 | 600 | -1 | 941 | 910 | 31 | AA | 707 | N3DUAA | LGA | DFW | 257 | 1389 | 6 | 0 | 2013 |
| 16 | 2013 | 1 | 1 | 559 | 559 | 0 | 702 | 706 | -4 | B6 | 1806 | N708JB | JFK | BOS | 44 | 187 | 5 | 59 | 2013 |
| 17 | 2013 | 1 | 1 | 559 | 600 | -1 | 854 | 902 | -8 | UA | 1187 | N76515 | EWR | LAS | 337 | 2227 | 6 | 0 | 2013 |
| 18 | 2013 | 1 | 1 | 600 | 600 | 0 | 851 | 858 | -7 | B6 | 371 | N595JB | LGA | FLL | 152 | 1076 | 6 | 0 | 2013 |
| 19 | 2013 | 1 | 1 | 600 | 600 | 0 | 837 | 825 | 12 | MQ | 4650 | N542MQ | LGA | ATL | 134 | 762 | 6 | 0 | 2013 |

Showing 1 to 20 of 1,000 entries

**Figura 3.35** – Visualización de la tabla en RStudio

Finalmente, filtraremos todos los vuelos según su retraso mediante funciones de Spark SQL y representaremos el gráfico de dichos retrasos con el paquete "*ggplot2*".

Una vez guardado el archivo en formato .R, mediante cualquier editor de texto, lo cargamos en RStudio (File/Open File) y ejecutamos el código de dicho archivo (si solo queremos seleccionar una parte, la seleccionamos y hacemos Control+Enter).

La representación del gráfico final es la siguiente:

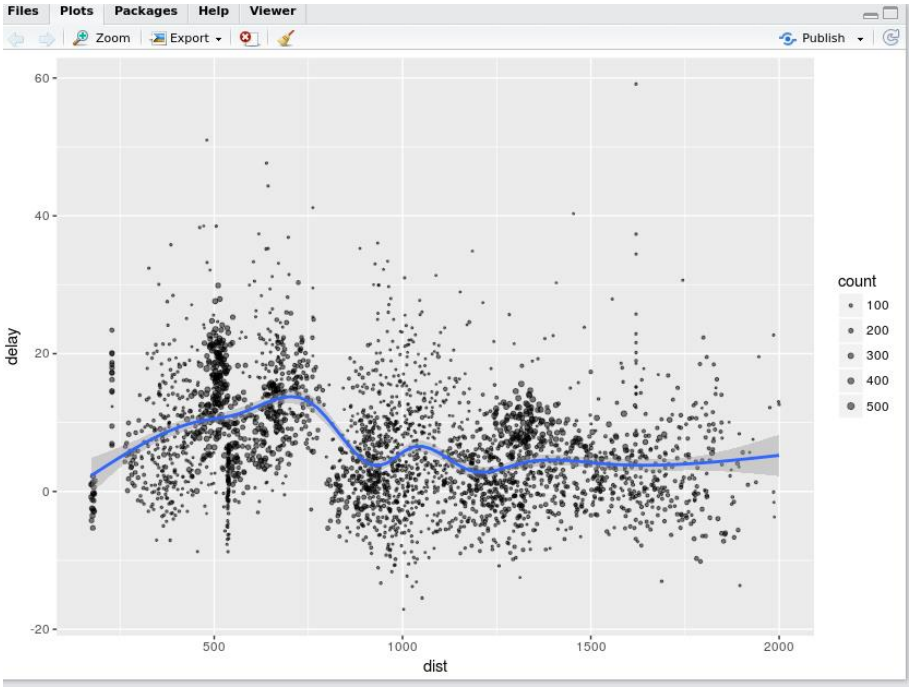



**Gráfico 3.36** – Representación de los retrasos de lo vuelos

Dónde podemos ver representado el retraso en el eje vertical y la distancia en el eje horizontal. La representación del gráfico se establece por puntos, donde el tamaño de dichos puntos nos indica cuántas veces ha sucedido.

### *3.3.1.1 - Análisis y conclusiones extraídas de los ejemplos con la librería dplyr*

Recomendamos el uso de esta librería para procesar datos, a poder ser, en forma de tabla y hacer un análisis estadístico de ellos y también el uso de la última versión de Spark (2.1.0 o 2.0.0) para que sea totalmente compatible con la librería y los paquetes. Creemos que es interesante para este tipo de análisis, por el contrario, no vemos que sea mucho más eficiente para extraer algún resultado más a parte de lo anteriormente comentado. Destacamos su fácil instalación y buena integración con R y Spark.

Dentro de la librería dplyr hay multitud de funciones a utilizar, podemos elegir y utilizar las que queramos en cada momento para procesar los datos de una manera u otra. En nuestros ejemplos hemos utilizado algunas tales como: SQL, Collect, Grouping, Piping, … Evidentemente se pueden utilizar otras e incluso más (incluso podemos editar datos en otros formatos de archivos importados), nosotros hemos utilizado algunas para ver el principal funcionamiento principal de la librería y el rendimiento que nos daba al ejecutar ejemplos más o menos sencillos.

## 3.3.2 - Machine Learning en R con Spark

En este apartado seguiremos usando la librería *Sparklyr* como base de todo (también lo combinaremos con el uso de la librería *dplyr* anteriormente utilizada), y nos centraremos en el uso de librerías y funciones relacionadas con Machine Learning (*spark.ml*), para tratar de hacer predicciones y estudios en forma de gráfico. En este apartado veremos el uso de Spark Machine Learning Library (MLlib) que es el básico con todas las funciones y el que te viene implementado de serie con Spark, y también miraremos por encima (sin tanta profundidad como el anterior) Sparkling Water (H2O) Machine Learning (adjuntado en el Anexo XIV), una librería que presenta todo lo de la primera, y que parece que está orientada a hacer cosas de más alto nivel, además de que se puede incluso utilizar como suite totalmente independiente sin necesidad de Spark o Hadoop para extraer resultados.

### *3.3.2.1 - Uso de Spark Machine Learning Library (MLlib)*

Esta librería presenta multitud de algoritmos a utilizar y analizar, intentaremos explicarlos brevemente y hacer algunos ejemplos de algunos para ver su funcionamiento.

Las funciones y algoritmos que se utilizan son los siguientes:

| Función / Algoritmo de ML | Descripción breve |
|---|---|
| ml_kmeans | K-Means Clustering |
| ml_linear_regression | Regresión Lineal |
| ml_logistic_regression | Regresión Logística |
| ml_survival_regression | Regresión de Supervivencia |
| ml_generalized_linear_regression | Regresión Lineal Generalizada |
| ml_decision_tree | Árbol de decisión |
| ml_random_forest | Bosque aleatorio (Random Forest) |
| ml_gradient_boosted_trees | Árboles con Gradiente |
| ml_pca | Análisis de Componentes Principales |
| ml_naive_bayes | Naive-Bayes |
| ml_multilayer_perceptron | Multilayer Perceptron |
| ml_lda | Latent Dirichlet Allocation |
| ml_one_vs_rest | Uno contra el Resto |

**Tabla 3.37** – Funciones y algoritmos de Machine Learning

Hay que destacar también que, a parte de los algoritmos, también existen transformadores de datos, utilidades y otras extensiones. Nosotros nos hemos centrado en los algoritmos que podemos utilizar principalmente, ya que queremos ver el funcionamiento general y las características que nos ofrece esta librería de Machine Learning (algunos añadidos los utilizaremos también).

En el siguiente apartado explicaremos brevemente algunas de las funciones y algoritmos anteriormente citados para podernos hacer una idea que herramientas hay y que podemos hacer con ellas, y para no hacerlo muy largo

realizaremos un ejemplo con muchos algoritmos de Machine Learning superpuestos para establecer una comparativa. Los otros ejemplos analizados individualmente estarán adjuntados en el Anexo V.

*Ejemplo – Comparación de las funciones de Machine Learning*

Utilizando la librería Sparklyr podemos utilizar una variedad de funciones y algoritmos de Machine Learning muy extensa, como ya hemos visto y comentado anteriormente. Este ejemplo (código íntegro adjuntado en el Anexo), trataremos de comparar el rendimiento de seis modelos de funciones en Apache Spark utilizando unos datos determinados.

Los datos que utilizaremos pueden predecir la supervivencia gracias a factores como la clase, el sexo, la edad y la familia.

En la primera parte del código gestionaremos los datos a través de funciones de Spark SQL (a través de la librería dplyr) y funciones de Spark ML.

En la segunda parte del código, utilizaremos diferentes algoritmos de Machine Learning para ver la predicción de cada uno. Para ello utilizaremos el modelo de supervivencia en función de varios predictores y sacamos los resultados con diferentes algoritmos de ML, tal y como muestra la siguiente Figura 3.38:

```
# Modelo de supervivencia en función de varios predictores
ml_formula <- formula(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked + Family_Sizes)

# Modelo de Regresión Logística
(ml_log <- ml_logistic_regression(train_tbl, ml_formula))

## Modelo de Árbol de decisión (Decision Tree)
ml_dt <- ml_decision_tree(train_tbl, ml_formula)

## Modelo de Bosque aleatorio (Random Forest)
ml_rf <- ml_random_forest(train_tbl, ml_formula)

## Gradient Boosted Tree Model
ml_gbt <- ml_gradient_boosted_trees(train_tbl, ml_formula)

## Naive Bayes Model
ml_nb <- ml_naive_bayes(train_tbl, ml_formula)

## Modelo de Red neuronal (Neural Network)
ml_nn <- ml_multilayer_perceptron(train_tbl, ml_formula, layers = c(11,15,2))

# Agrupación de los diferentes modelos de ML en una lista
ml_models <- list(
  "Logistic" = ml_log,
  "Decision Tree" = ml_dt,
  "Random Forest" = ml_rf,
  "Gradient Boosted Trees" = ml_gbt,
  "Naive Bayes" = ml_nb,
  "Neural Net" = ml_nn
)
```

**Figura 3.38** – Utilización de diferentes algoritmos de Machine Learning agrupados en una lista

En la tercera parte del código, crearemos una función para puntuar los diferentes modelos y también utilizaremos otra función llamada “lift” que compara lo bien que el modelo predice la “supervivencia” en comparación con la suposición aleatoria, finalmente lo representaremos tal y como muestra el Gráfico 3.39.

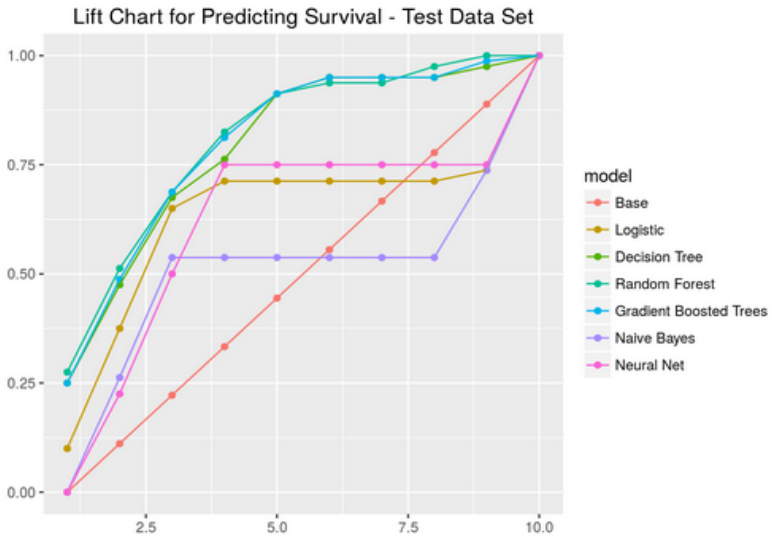



**Gráfico 3.39** – Resultado con todos los algoritmos de ML superpuestos

Como podemos observar en el gráfico, los modelos basados en árboles (Random Forest, Gradient Boosted Trees y Decision Tree) proporcionan las mejores predicciones frente a los demás.

Para acabar de hacer una comparativa con los diferentes modelos estudiados en este ejemplo, nos basaremos en la precisión de la predicción frente al AUC (siglas en Inglés: Area Under the Curve) que nos permite capturar el rendimiento para ciertos valores específicos, es decir, es la relación entre la confianza y el porcentaje de acierto obtenido que tiene que ser proporcional y cuanto más alto mejor. Cuanto mayor sea el valor de AUC, mejor será la predicción y el modelo. En el Gráfico 3.40, podemos ver la comparativa entre precisión y AUC:

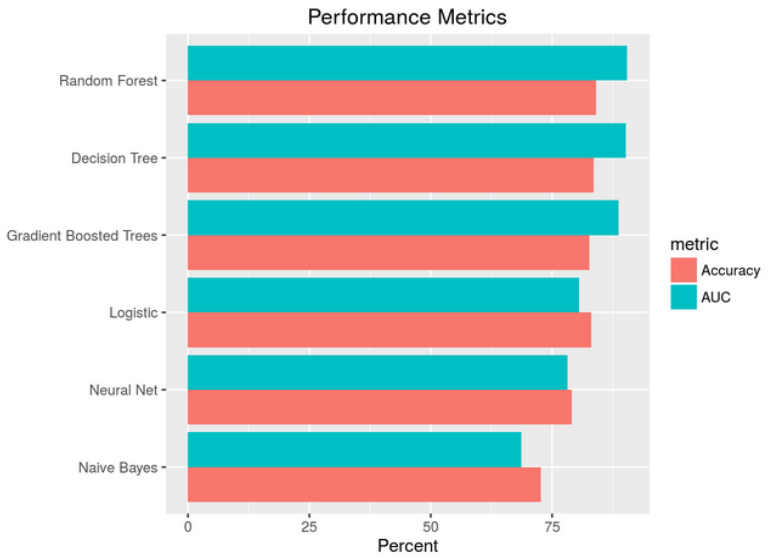



**Gráfico 3.40** – Comparativa de todos los modelos del ejemplo basándonos en la Precisión y AUC de cada uno

Tal y como vemos en el gráfico resultante, observamos que, por ejemplo, los modelos de árbol y de regresión logística tienen una precisión muy parecida, pero en cambio los primeros presentan un AUC mayor y por tanto se acercan más al resultado esperado y con mayor exactitud. Por tanto, para deducir que modelo es mejor que otro, nos tenemos que basar en la relación Precisión – AUC, y el que obtenga los valores más altos proporcionalmente será el modelo más adecuado a utilizar para el funcionamiento de ese ejemplo o código.

Hay otros dos análisis de los resultados que también podemos hacer, uno de ellos es que, dependiendo del modelo utilizado, se da más importancia a un atributo de estudio que a otro (por ejemplo, el modelo de Árbol de Decisión da más importancia al Sexo de las personas y el Gradient Boosted Trees da más importancia a la Edad, y a través de esto cada uno hace su predicción).
Y el otro, son los tiempos de procesado de cada modelo, si hacemos la prueba diferentes veces, podemos observar que modelos como el de Gradient Boosted Trees y el de Red Neuronal tardan más segundos que los otros métodos.

Finalmente, destacar que los modelos basados en árbol son los claros ganadores del ejemplo, aunque por ejemplo el Gradient Boosted Trees sea uno de los mejores, también nos lleva más tiempo de ejecución, en cambio, los otros 2 (Árbol de Decisión y Bosque Aleatorio) tienen un rendimiento bueno y un tiempo de ejecución rápido, están equilibrados.

#### *3.3.2.2 – Conclusiones de los ejemplos de Machine Learning en R (MLlib y H2O)*

- Como hemos podido observar en todos los ejemplos, Machine Learning en R nos ofrece una gran variedad de algoritmos y funciones para hacer predicciones. Elegiremos un algoritmo u otro (o una combinación de ellos) dependiendo de la clase de algoritmo del que se trate (clasificadores, regresión, detección de anomalías, etc) y de las características de cada uno (precisión, AUC, importancia de atributos de datos, rapidez de ejecución, etc).
- Dado que R nos proporciona resultados gráficos, creemos que junto a Machine Learning puede servir como herramienta muy útil para predecir resultados de otras tecnologías más complejas.

### 3.3.3 – Conclusiones generales de Apache Spark junto con el lenguaje R

En esta parte, haremos las conclusiones generales y finales del lenguaje R con Spark, después de todos los análisis de las librerías y de los ejemplos utilizados, destacaremos los aspectos positivos y negativos del uso de este lenguaje en el marco y ámbito de Big Data en términos generales. Por otra parte, también daremos nuestras recomendaciones.

Aspectos positivos:

- Conexión e integración entre R y Spark sencilla y sin problemas.

- Copia de datos y posterior tratado de datos con Spark SQL cómodo y eficaz.
- Datos procesados con funciones para extraer gráficos resultantes.
- Gráficos resultantes muy amigables, de rápido procesado y entendibles.
- El entorno gráfico te permite ver las variables en todo momento, tanto las de R como de Spark, para poder analizarlas y consultarlas. Además, los gráficos tienen un sitio reservado de aparición.
- Facilidad de programación en general.
- Multitud de paquetes a instalar y soporte para multitud de formatos de archivos.
- Flexibilidad de utilización de varios paquetes a la vez y alternativas para utilizar unos y otros (como en el caso de MLlib y H2O).
- Rapidez de ejecución y entorno amigable.
- Al ser de código abierto, muchos usuarios crean o editan librerías externas que también podemos utilizar y hacer uso.
- Es totalmente gratuito al tratarse de software libre “Open Source”.
- Gran soporte por parte de la comunidad con numerosas actualizaciones y documentación asociada.
- Fácil instalación en entornos GNU/Linux, y también gran soporte para otros SO como Microsoft Windows o Mac OS X.

Aspectos negativos:

- Conexión e integración entre R y Hadoop muy complicada, casi imposible.
- Los datos en R por defecto tienen un formato poco entendible.
- La ejecución de los procesos se produce en Spark y el resultado es devuelto y representado en R con el gráfico resultante.
- Solo y exclusivamente se pueden mostrar gráficos resultantes para analizar el estudio que hemos realizado, no se pueden hacer programas complejos más elaborados.
- La facilidad y flexibilidad de programación a veces se vuelve en contra, ya que a veces no coge bien las variables o no las procesa como debería.
- Algunos paquetes y librerías, dependiendo de las versiones de R y Spark utilizadas, no funcionan correctamente o no existen. Hay que tener extremo cuidado con las versiones porque puede no funcionar.
- Los mensajes de error que aparecen en consola a veces no son muy claros y no sabes exactamente porque falla y no funciona el programa.
- Difícil utilización para personas que no estén familiarizadas con entornos y lenguajes de programación ya que hay otras alternativas para sacar resultados estadísticos que no requieren este aprendizaje.

Recomendamos su uso para hacer análisis estadísticos y para extraer gráficos resultantes. Para hacer programas elaborados o diseñar aplicaciones de Big Data, aunque haya mucha presencia de paquetes, creemos que está un poco limitado, y por lo tanto, recomendaríamos el uso Hadoop o Spark con otros lenguajes (a poder ser los principales o nativos).

## 3.4 – Comparativa entre tecnologías y lenguajes de programación

Una de las partes más relevantes de este proyecto consiste en la elección de tecnologías y lenguajes de programación en el mundo actual de Big Data. Las tecnologías más usadas actualmente son Apache Hadoop y Apache Spark, la decisión de elegir una u otra ha tenido un campo de exploración detrás relevante, ya que hemos explorado todas las ventajas y desventajas de cada una, y hemos realizado una conclusión final para su elección.
Hoy en día, las grandes empresas utilizan una combinación de ambos para procesar las grandes masas de datos, ya que muchas no disponen de software propio y utilizan herramientas Open Source. Estas 2 herramientas son muy utilizadas dentro del sector. Hay mucha información al respecto por la red, aunque hay que saber que filtrarla adecuadamente y aplicarla como toca para ver las particularidades de dichas tecnologías.
Cabe destacar que dicha elección no significa que una tecnología sea mejor que la otra, si no que son totalmente complementarias, ya que una deriva de la otra y tienen diferentes capacidades y características. Además, la utilización de cada una también va a gustos personales y según para que vayamos a aplicarla. Aquí haremos una síntesis de todas los conceptos y características explicadas en este mismo capítulo.
La elección de lenguajes de programación, para evitar problemas con versiones y librerías, hemos elegido las nativas y principales en las que están escritas originalmente cada tecnología, más concretamente Java para Hadoop y Scala para Spark.

### 3.4.1 – Comparativa de características

A continuación, realizaremos la comparativa centrándonos en diferentes aspectos y características de cada uno.

#### *3.4.1.1 - Tiempo de procesado y rendimiento*

En términos generales, hemos comprobado que Apache Spark es más rápido que Hadoop debido a que procesa datos en memoria, en cambio, Hadoop precisa de un acceso a disco después de cada proceso MapReduce.
La principal ventaja es que Spark es bastante más rápido que Hadoop, pero como contrapartida requerimos de mucha más memoria, tal y como vemos en el Gráfico 3.41.

La comparación si se realiza con el mismo conjunto de datos, como hemos visto en este mismo capítulo, Spark sale ganando, aunque cuando dichos datos no caben en memoria, los resultados de tiempo de procesado son muy parecidos.

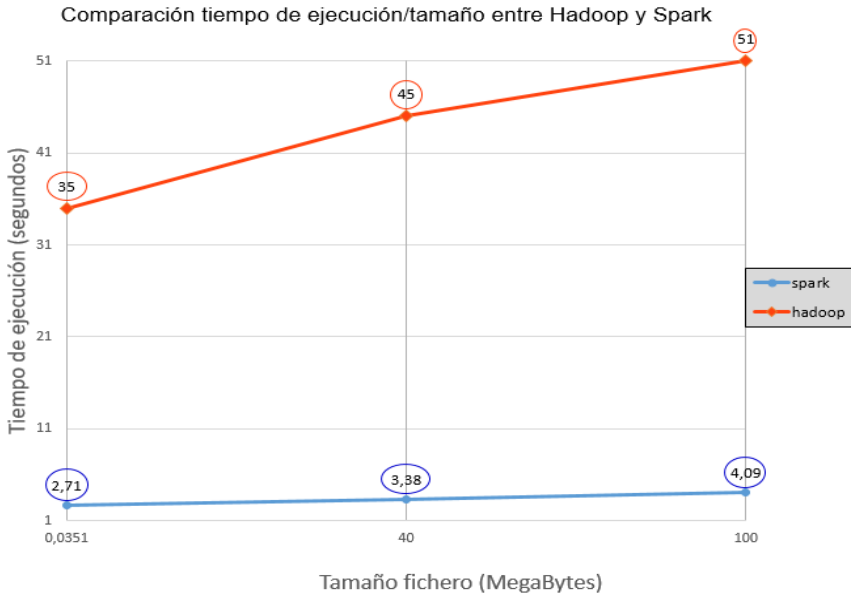



**Gráfico 3.41** – Comparativa entre Hadoop y Spark

El tiempo de procesado de R junto con Spark, en nuestro caso, utilizándolo en RStudio, solo hemos visto que con algunos paquetes tardaba un poco en instalarlos, pero en general, tenía un tiempo de procesado bastante rápido. El tiempo va asociado directamente a cuantas líneas de código hay en el programa, contra más líneas más tiempo requiere, aunque aconsejamos no ejecutarlo todo de golpe porque a veces presenta problemas, ejecutarlo por bloques y poco a poco para salvar problemas futuros.

### *3.4.1.2 - Coste y especificaciones*

Tanto Spark, Hadoop como el lenguaje R son software libre y por tanto accesible para todo el mundo y soportado por todos los sistemas operativos actuales.
Las especificaciones de hardware para cada uno son muy similares, aunque destacamos que Spark al usarse en memoria, requiere más recursos que por ejemplo Hadoop que precisa discos físicos para su almacenamiento respecto al despliegue de estructuras.

Para R las especificaciones tampoco son muy altas, ya que podemos también utilizarlo por consola y no requiere muchos recursos. En cambio, si utilizamos un IDE los recursos necesarios aumentan un poco, pero realmente no necesitamos grandes especificaciones para utilizarlo sin ningún tipo de problema.

Con la aparición de los sistemas en la Nube y los entornos virtualizados que dan muchos servicios de Big Data, suelen estar implementadas tanto las tecnologías de Hadoop como Spark, y podemos elegir cuál utilizar. También muchas soportan lenguaje R y puedes establecer gráficos resultantes.

#### *3.4.1.3 - Facilidad de uso y flexibilidad*

En este apartado destacamos el uso de Spark, ya que las APIs que podemos encontrar soportan multitud de lenguajes de programación, entre los cuáles están Java, Scala, Python y R. Tiene un gran soporte de librerías y por parte de la comunidad, para poder realizar diferentes programas además de la interacción por línea de comandos que también soporta todos esos lenguajes y tiene una gran facilidad de uso.

La utilización de R tiene que estar integrada y ligada con Spark, ya que no tiene soporte por parte de Hadoop, también cabe destacar la importancia de paquetes específicos para hacer la conexión entre ellos mucho más amena y amigable, además de los propios paquetes asociados en R para Machine Learning por ejemplo.

Por otro lado, Hadoop está nativamente desarrollado en Java, aunque también es capaz de soportar otros lenguajes de programación como C. Destaca principalmente la dificultad de programar con él, aunque presenta herramientas como hemos visto en este mismo capítulo como Hive, Impala o Pig que resultan de gran ayuda para realizar diferentes programas y disminuyen en gran medida su dificultad.

#### *3.4.1.4 - Instalación del entorno*

Cabe destacar que la instalación de Spark y Hadoop, dependiendo de la versión que escojamos va a variar considerablemente entre dichas versiones.
En esta parte destacamos la instalación de la versión 2.1.0 de Spark (la que hemos instalado nosotros) en Ubuntu 16.04 LTS, donde siguiendo unos cuantos pasos la instalación no consta de grandes complicaciones si estamos acostumbrados a trabajar en entornos GNU/Linux.

La instalación de R tampoco tiene gran complicación, la propia página web te marca los pasos para dicha instalación y supone una instalación corta y rápida.

Por otra parte, la instalación de Hadoop si que presenta bastantes complicaciones asociadas, dependiendo del modo de uso que elijamos, la instalación aún se puede complicar más, además de ser una instalación larga los pasos no son nada triviales. Hemos realizado una instalación de Hadoop nativa en Ubuntu 16.04 LTS para probar ejemplos sencillos en modo single node, y para la utilización de herramientas externas (como Hive, Impala o Pig) hemos utilizado entornos virtualizados como Cloudera.

En este aspecto, también declaramos Spark como justo vencedor, debido a la facilidad de instalación en comparación con Hadoop.

#### *3.4.1.5 – Compatibilidad*

Tanto Spark como Hadoop destacan por su gran compatibilidad con diferentes tipos de datos y diversas fuentes de los mismos.

Por otra parte, Spark puede llevar a cabo tanto procesamiento en tiempo real como procesamiento por lotes, mientras que Hadoop solamente por lotes.

R procesa diferentes tipos de archivos como CSVs para ser ejecutados en el programa.

#### *3.4.1.6 - Tolerancia a errores*

El uso de lenguaje R presenta una buena tolerancia a errores ya que te indica explícitamente por consola de que error se trata y como se puede solucionar, aún así hay algunos errores de declaraciones de variables que no los acaba de coger del todo bien, y te muestra un resultado que no es el esperado a veces. A pesar de eso, podemos decir que R tiene una buena tolerancia a errores.

Centrándonos en Spark y Hadoop, ambas presentan una buena tolerancia a fallos, aunque hay que destacar que Hadoop presenta una mayor tolerancia gracias al sistema de replicación que implementa por defecto, que hace que cuando caiga un nodo o un conjunto de ellos, siga funcionando correctamente debido a que hay copias de backup suficientes para que funcione.

Esta última característica de Hadoop no hemos podido comprobarla totalmente en acción, ya que la instalación nativa que hemos realizado era en modo single node y en el sistema Cloudera tampoco se podía apreciar del todo esta característica.

#### *3.4.1.7 - Parámetros de Seguridad*

Apache Spark presenta autenticación por clave compartida. Hadoop en cambio puede ejecutarse sobre YARN y hacer uso del sistema de archivos HDFS, así como su autenticación, comprobar los permisos de los ficheros HDFS y encriptación entre nodos.
En este apartado, la seguridad de Hadoop es más completa que la de Spark.

#### *3.4.1.8 – Lenguajes principales y librerías*

Hadoop utiliza como lenguaje nativo por defecto Java y no tiene ninguna librería presente para hacer más funciones más allá de MapReduce, exceptuando las herramientas externas que ya hemos comentado anteriormente.
Spark utiliza como lenguaje principal Scala y presenta multitud de librerías disponibles para hacer diferentes funciones además del soporte y desarrollo de la comunidad.

R es un lenguaje que tiene multitud de paquetes a instalar disponibles para asociarse con Spark y poder hacer diferentes ejemplos para conseguir gráficos resultantes.

### 3.4.2 – Resumen y elección final

En esta parte final, a modo de resumen haremos una tabla con las características anteriormente comentadas y finalmente extraeremos conclusiones sobre la elección de tecnología y lenguaje de programación que vamos a utilizar.

| | **Tecnologías** | |
|---|---|---|
| **Evaluación y criterios** |  |  |
| Tiempo de procesado y ejecución | Procesado basado en disco y más lento | Procesado basado en memoria y rápido |
| Coste y especificaciones | Software libre, requiere menos recursos | Software libre, requiere más recursos. |
| Facilidad de uso y flexibilidad | Compatible con Java o C<br><br>Herramientas externas<br><br>Dificultad de programación | Compatible con Java, Scala, R y Python<br><br>Multitud de soporte de librerías<br><br>Facilidad de programación |
| Instalación del entorno | Instalación larga y compleja | Instalación corta y rápida |
| Compatibilidad | Tipos de datos | Tipos de datos |
| Tolerancia a errores | Replicación de información, copias de backup | Buena |
| Parámetros de seguridad | YARN y HDFS | Clave compartida, HDFS |
| Lenguaje principal | Java | Scala |
| Librerías integradas | Ninguna | Spark SQL,<br>Spark Streaming, MLlib, GraphX |

**Tabla 3.42** – Comparativa a modo de resumen entre Apache Hadoop y Apache Spark.

Después de analizar todos los puntos y exponerlos en forma de tabla, el tiempo de procesado y ejecución y la facilidad de uso y flexibilidad hacen que nuestra elección sobre que tecnología usar sea clara: ***elegimos Spark***. A pesar que requiere una gran cantidad de memoria, creemos que las ventajas son mucho mayores que las desventajas y es por eso que nos decantamos por él.

Además, creemos que Spark es una tecnología mucho más moderna y adaptada a los últimos tiempos con gran soporte de librerías hacia campos actuales como

Streaming o Machine Learning, en cambio Hadoop es bastante más antiguo y está mucho más limitado, aunque se sigue utilizando.

En este apartado también haremos un análisis de los lenguajes de programación disponibles con tal de realizar la elección para utilizar conjuntamente con Spark, ya que es nuestra tecnología elegida.

| | **Lenguajes de programación** | | | |
|---|---|---|---|---|
| **Evaluación y criterios** |  |  |  |  |
| Compatibilidad | Hadoop y Spark | Spark | Spark | Spark |
| Lenguaje nativo/principal | Nativo para Hadoop | Principal para Spark | No nativo | No nativo |
| Grado de dificultad (de 0 al 10) | 8 | 7 | 6 | 6 |
| Soporte para librerías y paquetes | Muchas librerías, pero sin poder utilizarse en Hadoop | Multitud de librerías propias y también utiliza las de Java | Multitud de paquetes a utilizar | Muchas librerías presentes y versátiles |
| Nuevo para el estudiante | NO | SÍ | SÍ | SÍ |
| Curva de aprendizaje | Difícil de entender<br><br>(Dificultad exponencial) | Mayor complejidad para tareas específicas<br><br>(Dificultad logarítmica) | Dificultad lineal | No utilizado |

**Tabla 3.43** – Comparativa entre los diferentes lenguajes de programación

Basándonos en el uso de los diferentes lenguajes que hemos utilizado en este mismo capítulo (exceptuando Python), tras realizar la Tabla 3.43, concluimos que, ***a nuestro parecer, el mejor lenguaje para utilizar conjuntamente con Spark es Scala***, debido a que es el lenguaje principal que utiliza, el que tiene mayor soporte y que es un lenguaje totalmente nuevo para nosotros.

También destacamos la utilización de R para hacer diferentes análisis estadísticos de diferentes datos, muy similar a la utilización de Matlab y completamente nuevo para nosotros. Aunque R destaca por sus funciones

gráficas, hemos descartado su uso final ya que para elaborar programas complejos no nos da tantas opciones como puede ser Scala, aún así hemos evaluado las posibilidades que nos ofrecía en este mismo capítulo.

# CAPÍTULO 4: DESARROLLO EXPERIMENTAL EN APACHE SPARK

Para el capítulo más práctico, nuestra tecnología escogida es Spark, más específicamente nos gustaría evaluar sobre las posibilidades que ofrece Machine Learning, sus diferentes algoritmos y finalmente reproducir un experimento con las herramientas que más nos convencen.

## 4.1 - Escenario inicial

Tras buscar y analizar diferentes opciones, nos hemos decantado por realizar diferentes predicciones de la posición de la gente en un entorno, según la potencia recibida por múltiples Acces Points a los que se conectan con sus dispositivos. Para empezar a trabajar disponemos de un dataset realizado en la UJI [25] que contiene la siguiente información:

- 520 columnas con la potencia recibida por cada Access Point en dBs
- Longitud del dispositivo en el lugar de toma de medidas
- Latitud del dispositivo en el lugar de toma de medidas
- Edificio (del 0 al 2)
- Planta (de la 0 a la 4)
- Space ID (un identificador unívoco para cada espacio)

Otras columnas que hemos obviado durante este proceso contienen información como el identificador del dispositivo, un ID para la persona que toma las medidas o un timestamp de cuando se tomaron las medidas.

## 4.2 - Etapa de exploración

Inicialmente nos dedicaremos a ver que opciones tenemos, como vamos a usar Spark, lo más inmediato es ir a ver que ofrece la librería MLlib.

Anteriormente ya hemos visto un apartado sobre esta librería, pero no hemos profundizado en ella. MLlib presenta 3 características muy interesantes, la posibilidad de trabajar con un formato de archivo que la mayoría de algoritmos ya entienden (LIBSVM), muchas funciones para procesar los datasets y ajustarlos a nuestras necesidades y finalmente una multitud de algoritmos con los que trabajar.

### 4.2.1 - Features de MLlib

Vamos a ver que implica y porque son tan importantes los 3 aspectos comentados.

#### *4.2.1.1 - Formato libsvm*

LIBSVM es el acrónimo de Library for Support Vector Machines, a nosotros nos interesa el formato de datos que se diseñó con esta librería.


**Figura 4.1** – Ejemplo de archivo de texto en formato LIBSVM

Este formato es aceptado por casi todos los algoritmos de Machine Learning así que nos será fundamental tener el dataset en este formato. Cada fila contiene un label (lo que queremos predecir) al principio y todos los features numerados separados por dos puntos.

#### *4.2.1.2 - Pre y post procesado de datos*

La librería de Machine Learning de Spark aporta varias funcionalidades para procesar los datos de un dataset, cambiar sus valores, extraer datos concretos, etc. Algunas de estas funcionalidades se basan en simples ejecuciones de SQL y sobre estas no comentaremos nada, aunque en nuestro código aparezcan, otras como la normalización si tendrán una importancia mayor ya que nos ayudaran a facilitar el trabajo a los algoritmos a la hora de leer los datos.

Es importante destacar que Spark presenta una desventaja y es la retrocompatibilidad, en algunos cambios de versiones vemos como la librería “mllib” pasa a ser “ml” y aunque esto debería estar solventado con tan solo cambiar el nombre, las dos coexisten a la vez y aunque tienen funciones similares no son precisamente iguales.

#### *4.2.1.3 - Algoritmos*

Durante el capítulo usaremos diversos algoritmos de Machine Learning, vamos a realizar una pequeña introducción sobre ello.

El concepto de Machine Learning no es nuevo, pero está viviendo un renacimiento gracias a la mayor disponibilidad de datos y capacidad de cómputo. Machine Learning es, a groso modo, una serie de reglas abstractas que permiten a la computadora aprender por si sola sin tener que ser programada. Los algoritmos juegan un papel muy importante en esta era del Big Data, ya que nos permiten predecir lo que pasará.

Según que queramos predecir diferenciamos dos tipos de problemas; de regresión, donde la variable es un valor numérico, o problemas de clasificación donde la variable es un conjunto de estados discretos (un “string”).

Una vez identificado el tipo de problema, podemos aplicar algoritmos que se clasifican en 3 grupos:

- Modelos lineales: encuentran una línea que se ajuste a una nube de puntos.
- Modelos de árbol: construyen unas reglas de decisión que se pueden representar como las ramas de un árbol.
- Redes Neuronales: replican el comportamiento del cerebro.

Una vez repasados los apartados importantes de la librería de Machine Learning, vamos a empezar a aplicar diferentes algoritmos al dataset que hemos comentado al inicio del capítulo.

### 4.2.2 - Prueba 1: Decision Tree

La primera prueba que haremos será ejecutar un Decision Tree sobre nuestro dataset, al ser la primera prueba vamos a explicar un poco sobre como se ejecutan los algoritmos de ML.

Primero hay que preparar el dataset que usaremos, en nuestro caso hemos decidido empezar eliminando todos los labels menos uno, concretamente el label que usaremos es una fusión del número de edificio, planta y SpaceID, ya que el modelo de Decision Tree es un clasificador. Tras convertirlo del formato original (CSV) a LIBSVM con un script de Python, podemos empezar con nuestro código. En todas las pruebas habrá una parte común en la que crearemos un contexto (o sesión según la librería que usemos dentro de MLlib de Spark), cargaremos los datos y los dividiremos al azar en dos partes, Training y Test. Training servirá para entrenar el algortimo y Test se usará al final para comprobar como de efectivo es.

```
val spark = SparkSession.builder.appName("localitzador").getOrCreate()
Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

/.../

...
val data = spark.read.format("libsvm").load("/home/jpf/Downloads/wifiprimitiu/csv/proba")
```

**Figura 4.2** – Código para iniciar sesión en Spark

```
// Dividir les dades en training and test sets (70/30)
val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))
```

**Figura 4.3** – División en múltiples Datasets

Para realizar la prueba con Decision Tree hemos usado usado una función de pipeline, no vamos a entrar en detalles ya que no la usaremos siempre, pero el pipeline nos permite ejecutar diversas operaciones concatenadas. Como vemos en la Figura 4.4 crear un Decision Tree es muy sencillo, en general los algoritmos

básicos no son difíciles de crear, pero según si tiene o no variables puede ser más largo.

```
val dt = new DecisionTreeClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
```

**Figura 4.4** – Constructor del modelo

Tras ejecutar el código pedimos que nos pase por pantalla 3 cosas, una pequeña tabla con 5 predicciones, el error cometido y el árbol que ha usado.

```
+--------------+------+--------------------+
|predictedLabel| label|            features|
+--------------+------+--------------------+
|       32136.0|1009.0|(520,[0,1,2,3,4,5...|
|       32136.0|1009.0|(520,[0,1,2,3,4,5...|
|       32136.0|1009.0|(520,[0,1,2,3,4,5...|
|       32136.0|1010.0|(520,[0,1,2,3,4,5...|
|       32136.0|1010.0|(520,[0,1,2,3,4,5...|
+--------------+------+--------------------+
only showing top 5 rows

Test Error = 0.9444260119442601
```

**Figura 4.5** – Predicciones y Test Error

En la Figura 4.5 podemos observar que la columna predictedLabel no tiene nada que ver con la columna label y vemos que el error de predicción es de un 94.4%.

Vamos a ver como realiza la predicción el árbol de decisión para entender mejor el porque de este resultado tan nefasto.

```
Learned classification tree model:
DecisionTreeClassificationModel (uid=dtc_7add1f49141b) of depth 5 with 49 nodes
  If (feature 104 <= -1.0)
   If (feature 22 <= -63.0)
    Predict: 0.0
   Else (feature 22 > -63.0)
```

**Figura 4.6** – Model del Decision Tree

Vemo en la Figura 4.6 que el modelo ha decidido que el feature (access point) 104 es crítico y que, por lo tanto, si su valor es mayor de -1 dB se decantará por una rama y si es menor por otra.

Es importante tener en cuenta que contamos con 520 access points repartidos en 3 edificios, si la primera clasificación es errónea, lo serán todas las siguientes y vemos que es justamente lo que ocurre.

Si sumamos esto al hecho que el modelo ha tardado 335 segundos vemos que para nuestro objetivo un Decision Tree no es apropiado, este modelo serviría para un dataset con menos features, para un trabajo más complejo se podría usar un Random Forest (varios Random Tree). Hay que destacar que el tiempo indicado es el tiempo total de construcción del modelo, entrenamiento y

finalmente predicción, la predicción tan solo tarda 13,95 segundos, de todos modos, este tiempo junto a la precisión sigue haciendo que este algoritmo no sea bueno para nuestro objetivo.

### 4.2.3 - Prueba 2: Logistic Regression

Los siguientes algoritmos que hemos decidido probar son SVM y Logistic Regression como algoritmos de regresión en contrapartida al árbol de decisión que es un clasificador.
No dedicaremos mucho tiempo a esta prueba ya que tras probar algoritmos como "linear regression" o SVM, no obteníamos resultados o no eran los que esperábamos. En la Figura 4.7 observamos algunos de los datos que nos devuelve el programa, el RMSE que obtenemos es de un 70% lo cual mejora respecto a la experiencia previa con el clasificador "Random Tree".

```
9,0.004192300302237953] Intercept: 10030.977209077786
numIterations: 11
objectiveHistory: [0.4999999999997726,0.3524855611443946,0.1012001674612627,0.084113
```

**Figura 4.7** – Parte resultado Logistic Regression

Como hemos comentado los algoritmos de regresión presentan unos resultados extraños, ya que solo enseñan métricas del modelo entrenado, en ningún caso se usan para predecir, queda por lo tanto descartado su uso para posteriores pruebas.

### 4.2.4 - Prueba 3: Naive Bayes

El algoritmo de Naive Bayes es un clasificador probabilístico donde los features se pueden presentar en formato binario o en "múltiples clases" como es nuestro caso.

Bayes presenta un requisito y es que los datos del dataset tienen que ser positivos. Nuestro dataset está formado por potencias en dBs en rangos negativos, para agilizar el proceso y dado que el algoritmo funciona al margen del tipo de datos, vamos a pasar las potencias a valor absoluto.

```
val data = MLUtils.loadLibSVMFile(sc, "/home/jpf/Downloads/bayes1.1/bayes")

// Separar data en training (60%) i test (40%).
val Array(training, test) = data.randomSplit(Array(0.6, 0.4))

val model = NaiveBayes.train(training, lambda = 1.0, modelType = "multinomial")

val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
```

**Figura 4.8** – Declaración modelo Bayes y aplicación

Como vemos en la Figura 4.8 la aplicación de este algoritmo es igual de sencilla que los vistos anteriormente, aunque esta vez tenemos algunas peculiaridades como el tipo de modelo o el factor de "smoothing" alpha.

En la Figura 4.9 observamos que los primeros resultados son poco prometedores, pero mejoran con respecto a otras pruebas.

```
accuracy0.1366344005956813
Temps=12.533613144s
```

**Figura 4.9** – Precisión y tiempo de ejecución

Este modelo básico nos da una precisión del 13%, y vemos que tarda 12 segundos. Esto supone una leve mejora respecto al modelo de Decision Tree y una mayor eficiencia en cuanto a tiempo.

### 4.2.5 - Prueba 4: K-NN

Como última prueba queremos usar un modelo puramente memorístico, pero curiosamente MLlib de Spark no aporta ningún algoritmo de este tipo, así que hemos tenido que buscar un paquete externo de un tercero para poder usar K-Nearest Neighbours, ya que no se trata de un algoritmo de ML propiamente dicho. Su instalación no nos ha resultado especialmente fácil, aunque eso también se debe a la poca experiencia que tenemos con este entorno, en el Anexo X dedicaremos un pequeño apartado a su explicación.

Este algoritmo tiene dos modalidades, clasificación o regresión, para esta prueba, y porque hemos usado el dataset cuyo label es el SpaceID, veremos el modelo de clasificación

```
val knn = new KNNClassifier()
  .setTopTreeSize(dataset.count().toInt / 500)
  .setK(10)
  .setPredictionCol("predicted")
```

**Figura4.10** - Declaración del classificador K-NN

Observamos en la Figura 4.10 que hay dos parámetros no vistos hasta ahora, TopTreeSize y K, en este caso no los hemos modificado.

Tanto declarar el modelo como usarlo es muy sencillo y con pocas líneas de código ya podemos obtener los primeros resultados como se muestra en la Figura 4.11

```
accuracy0.6516725798276736
Temps=112.573383348s
```

**Figura 4.11** – Precisión y tiempo de ejecución

Ambos números han subido respecto a lo que habíamos visto con Bayes, pese a que el modelo tarda 100 segundos más en ejecutarse vemos que la precisión es ahora de un 65%.

### 4.2.6 – Conclusiones Parciales

Hemos decidido comentar 4 pruebas diferentes que hemos realizado pero el trabajo detrás ha sido más extenso. Aunque el modelo de Bayes parezca a simple vista inviable, tras diversas pruebas con otros datasets hemos visto que realmente es una opción interesante. En la Tabla 4.12 podemos ver las diferencias entre los dos modelos más prometedores.

| *Label* | **Modelo naive Bayes** | | **Modelo K-NN regresión** | | **Modelo K-NN clasificación** | |
|---|---|---|---|---|---|---|
| | Precisión | Tiempo(s) | Precisión | Tiempo(s) | Precisión | Tiempo(s) |
| Edificio | 0.977 | 9.45 | 0.991 | 40.27 | 0.997 | 42.37 |
| Planta | 0.755 | 11.04 | 0.959 | 43.77 | 0.989 | 41.85 |
| SpaceID | 0.141 | 12.53 | 0.097 | 46.75 | 0.699 | 43.28 |
| Longitud | 0.457 | 13.38 | 0.026 | 53.39 | N/A | N/A |

**Tabla 4.12** – Comparación de tiempo y precisión entre Bayes y K-NN

De esta tabla podemos extraer algunas conclusiones muy interesantes. Por un lado, podemos ver que el modelo de clasificación de K-NN es mejor que el de regresión para los labels que usamos, esto es bastante lógico si tenemos en cuenta que edificio tiene 3 números posibles y planta 4, no es un modelo binario pero se acerca. También es importante destacar que la dificultad para predecir aumenta según la cantidad de posibilidades, a medida que bajamos en la tabla, los labels pueden tomar más valores y eso se ve reflejado en la precisión. El modelo de clasificación no es capaz de ejecutarse con la longitud (al ser un paquete externo a Spark no hemos indagado mucho en ello) pero curiosamente Bayes (tomado como modelo de clasificación según Spark) si es capaz de funcionar con ella pese a tener prácticamente 1.000 posibles valores y aún así obtener una precisión de prácticamente el 50%.

Podemos observar también que SpaceID muestra un comportamiento errático con el modelo de Bayes, y que, en el modelo de regresión tambien baja mucho su precisión pese a ser un número con muchos menos dígitos al de la longitud. La explicación más lógica para este suceso, es que, al ser un número construido por nosotros, los valores no están claramente definidos, dando como resultado dos localizaciones con potencias muy diferentes que pueden tener valores muy similares.

Con excepción de la longitud, los números nos indican que K-NN es un modelo mucho más preciso, pero sabemos que requiere de mucho espacio para almacenar todo el set de datos de entrenamiento y vemos que los tiempos de ejecución son sustancialmente más altos (todos los tiempos engloban tanto la construcción del modelo como la predicción). Si solamente nos preocupara eso, podríamos intentar guardar el modelo entrenado de K-NN para usarlo más tarde pero el paquete para Spark no dispone de esta opción.

## 4.3 – Proyecto experimental

Si solo quisiéramos tener una idea aproximada de donde se encuentra un dispositivo, y el tiempo de evaluación no fuera un problema, utilizaríamos un modelo K-NN, pero queremos evaluar el potencial de una aplicación de este tipo, y, por tanto, necesitamos encontrar un equilibrio entre el tiempo y la precisión, para el resto de pruebas utilizaremos tanto K-NN como Bayes.

En adelante seguiremos el proyecto basándonos en un documento de la Universidad de Minho (Portugal) [26] que usan el mismo dataset que nosotros. En su caso, ellos usaron Matlab o una herramienta similar, y por lo tanto, la idea de Big Data no aparece por ningún lado, en cambio lo que haremos nosotros es abordar el problema de una manera similar a la suya, escalando el problema.

Inicialmente pretendíamos normalizar los valores con la función “MinMaxScaler” que nos permite fijar un rango en el que ajustar todos los valores disponibles, en nuestro caso será de 0 a 1, donde 0 será la potencia mínima recibida y 1 la máxima, pero tras múltiples intentos fallidos hemos tenido que conformarnos con la función “Normalizer” que deja al usuario pocas variables y por lo tanto hará la normalización como quiera, en este caso pasa los valores de -1 a 1, como esto es un problema, ya que Bayes no trabaja con números negativos, primero tendremos que obtener el valor absoluto.

Tras normalizar, empezaremos con Bayes, y vamos a evaluar diferentes parámetros de Alpha. En la Figura 4.13 no se aprecia ningún cambio en la precisión de medida, dependiendo del dataset de entrenamiento (recordamos que se escoge al azar a partir del dataset original) si se aprecia alguna diferencia, pero siempre es en el 4º o 5º decimal.

```
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 0.0
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 0.1
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 0.3
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 0.5
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 0.7
Precisió (validation) = 0.9786368260427264 pel model amb lambda = 1.0
```

**Figura 4.13** – Test parámetro Alpha

Tras evaluar el tiempo, con y sin medida del parámetro Alpha, vemos que este proceso tarda alrededor de 4 segundos lo cual, para lo que aporta, es demasiado; dejaremos Alpha con valor 1.

La primera predicción que haremos será sobre el edificio, más tarde encontraremos la planta y de allí las coordenadas. Vamos a ver si la predicción mejora tras normalizar.

```
accuracy:0.9802615346656798
```

**Figura 4.14** – Predicción del modelo sobre los edificios

Tras ver que hay una mejora mínima en la precisión nos planteamos como escalar las predicciones. Hay dos enfoques posibles:

- Preparar un dataset que contenga los access points, el edificio, la planta y las coordendas como features sin label (un label simbólico ya que será modificado más tarde). Con el edificio predicho, eliminar todos los datos que no correspondan a dicho edificio y predecir la planta. Por último, hacer lo mismo con las coordenadas.
- Tener una carpeta con 15 datasets, correspondientes a los 3 edificos y cada una de las plantas dentro de estos, y usar el correspondiente en cada caso a medida que hagamos predicciones.

Ambos métodos presentan ventajas e inconvenientes, así que intentaremos hacerlo de las dos maneras para ver cual presenta mejores resultados.

La primera alternativa, la que nos parece más limpia en cuanto a código y que ocupará menos espacio en disco (ya que tiene que guardar un solo dataset) nos ha sido imposible de realizar, el formato LIBSVM tiene un formato muy estricto y no hemos conseguido llegar a ello tras realizar las transformaciones pertinentes.

La segunda opción lleva más tiempo de trabajo al margen del código, ya que se tienen que preparar 15 archivos con los datos correspondientes. Tras estos preparativos, escribimos las condiciones en el código que harán que busquemos en un dataset u otro, simplemente se trata de encontrar el edificio más repetido, hacer lo mismo con el piso dentro del edificio correspondiente y finalmente encontrar las coordenadas.

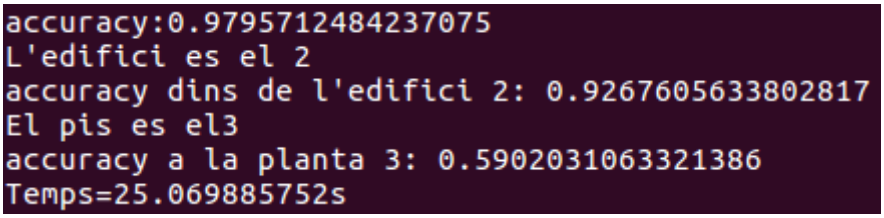



**Figura 4.15** - Opción 2, primer resultado con Bayes

En la Figura 4.15 vemos como escoge un edificio (el que más se repite) y dentro de este, con una precisión del 92%, escoge un piso. La precisión final (coordenadas) es del 60% y todo esto en 25 segundos. Respecto a la prueba inicial realizada con Bayes vemos que la precisión aumenta en un 15% a costa de 13 segundos, el doble de tiempo.

Ahora vamos a adaptarlo a K-NN, aunque parezca trivial, tiene otros requisitos y los resultados los devuelve en DataFrames, así que habrá que hacer más cambios de los que originalmente teníamos en mente.

En la Figura 4.16 con los resultados ya obtenidos, vemos que no difiere mucho de lo que esperábamos, el tiempo es mucho mayor que el obtenido con Bayes y aunque la precisión mejora respecto a realizarlo sin escalar, la eficacia de K-NN para calcular las coordenadas sigue siendo pobre.

```
accuracy0.9972568578553616
L'edifici es el 2
accuracy dins de l'edifici 2: 0.9934437543133195
El pis es el2
accuracy dins de la planta 2: 0.03333333333333333
Temps=72.489429377s
```

**Figura 4.16** – Opción 2, primeros resultados con K-NN

La primera predicción muestra unos resultados similares para ambos casos, la segunda y la tercera ya empiezan a diferir, vamos a hacer una fusión de ambos algoritmos. Usaremos Bayes para predecir el edificio, ya que la mejora de K-NN es mínima y tarda mucho tiempo, y K-NN para el piso, ya que el dataset es menor y eleva la precisión hasta casi el 100% y finalmente Bayes para las coordenadas.

```
accuracy:0.9774025329029054
L'edifici es el 2
accuracy dins de l'edifici 2: 0.9964144854786662
El pis es el3
accuracy a la planta 3: 0.6132665832290363
Temps=54.499904787s
```

**Figura 4.17** – Opción 2, resultados Bayes + K-NN

El tiempo que añade de más K-NN es considerable, vamos a resumir los resultados en los siguientes Gráficos.

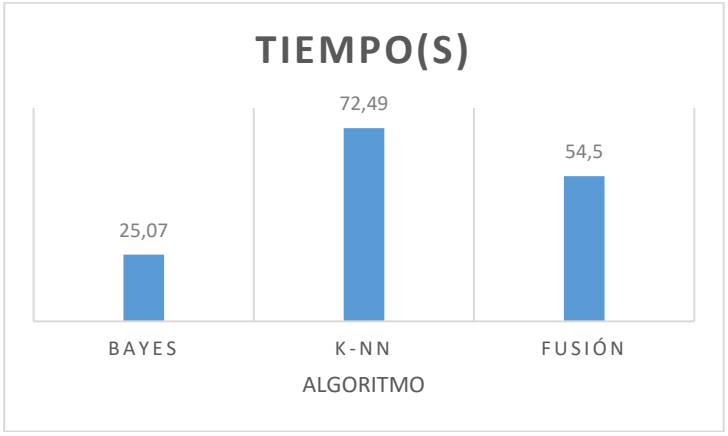



**Gráfico 4.18** – Comparación del tiempo de ejecución

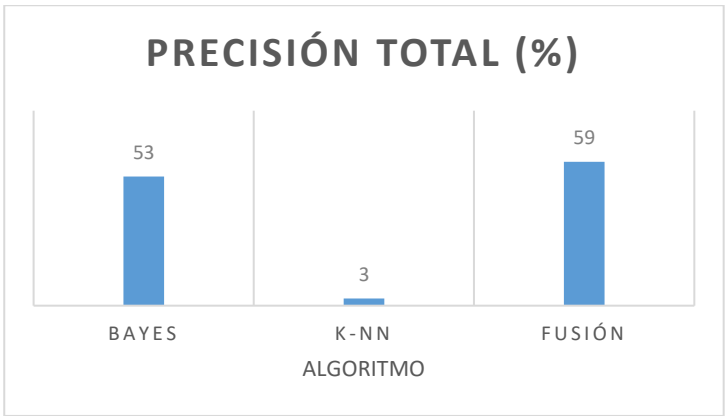



**Gráfico 4.19** – Comparación de la precisión

La precisión total es el producto de las precisiones parciales y nos indica la precisión final con la que obtenemos los resultados.

Como vemos en el Gráfico 4.19, la inexactitud de K-NN a la hora de encontrar las coordenadas hace que su precisión total baje mucho, y que, por lo tanto, quede descartado para un escenario de este estilo. En cuanto a la fusión de algoritmos, ganamos algo de precisión con ella, respecto al modelo solo con Bayes, pero el tiempo se duplica (Gráfico 4.18), lo que implica que, en según que aplicación, no nos puede interesar.

Es muy importante tener en cuenta que hemos usado el modelo de K-NN de clasificación en todos los pasos excepto en el último que hemos usado regresión. Bayes por su lado, en Spark, es un clasificador, así que finalmente lo que obtendremos siempre serán unas coordenadas preentrenadas, no nos equivocaremos por unos metros, sino que, las coordenadas estarán bien o habrá predicho otras totalmente diferentes.

Hemos realizado algunas pruebas solo con K-NN para ver que margen de mejora tiene según el porcentaje de los datos que dedicamos a entrenamiento, al tratarse de un modelo memorístico hemos supuesto que más entrenamiento daría resultados más precisos y en cualquier caso la precisión no se acerca al 3% que obteníamos con la estrategia de escalarlo. K-NN requiere de un tratado previo de los datos bastante más exhaustivo y de una buena elección de parámetros para que pueda ser considerado para una aplicación real (en cuanto a regresión). Se podrían seguir realizando más pruebas, además de lo anteriormente comentado, podríamos cambiar el valor de K, usar datasets muy específicos para comprobar la precisión en diferentes espacios, etc., pero para el objetivo de este capítulo que era investigar y trabajar con la librería de Machine Learning y reproducir el experimento de la Universidad de Minho, no lo haremos.

Uno de los objetivos que persiguen muchas empresas es la capacidad de obtener un modelo universal de posicionamiento en interiores, queda claro que es posible predecir la posición con bastante exactitud y en un tiempo tolerable, pero, y aunque lejos de una solución comercial, este modelo que hemos entrenado sirve tan solo para la UJI, la Universidad y el campus donde se han tomado las medidas previamente.

A modo de conclusión, vemos que la librería MLlib de Spark presenta unas herramientas muy potentes que pueden procesar cantidades ingentes de datos en poco tiempo (hay que tener en cuenta que estas pruebas se han realizado en un ordenador de sobremesa que no está enfocado al uso de herramientas Big Data) y que abre muchas puertas a la industria para poder darle un uso muy beneficioso a todos los datos que reciben a diario.

# CAPÍTULO 5: CONCLUSIONES

En este apartado mostraremos las conclusiones generales del proyecto ya que hasta ahora, solo hemos mostrado las conclusiones específicas respectivas de cada capítulo. En primer lugar, valoraremos los objetivos principales del proyecto y, seguidamente, expondremos las conclusiones. Por último, veremos las posibles perspectivas de futuro de este proyecto en diferentes ámbitos.

## 5.1 – Objetivos cumplidos

Durante la ejecución de este proyecto, surgen algunos problemas e inconvenientes que pueden obligar en cierta medida a modificar los objetivos iniciales. A continuación, haremos un recordatorio de los objetivos principales y en que medida se han cumplido.

- *Investigar que es Big Data, de que se trata este concepto y todo lo que hay detrás.*

Este objetivo era totalmente imprescindible, ya que se trataba de adentrarse y aprender todo lo posible sobre Big Data. Dicho objetivo, creemos que se ha cumplido sin complicaciones y con creces, ya que, a parte de ver las tendencias actuales y futuras, hemos podido ver su crecimiento totalmente exponencial tanto en empresas como en diferentes ámbitos, así como las principales utilidades Open Source más destacadas que lo hacen posible.

- *Estudiar y usar las diferentes tecnologías, herramientas Open Source y lenguajes de programación presentes hoy en día relacionados con Big Data. Realizar un análisis individual y comparativo, además de establecer una pequeña guía introductoria para gente que tenga conocimientos sobre programación, pero no sobre Big Data.*

Creemos que esta es la parte más importante y troncal del proyecto, y que, por tanto, se ha cumplido con creces, ya que se trataba de exponer la gran multitud de tecnologías y herramientas Open Source más importantes y utilizadas, y analizarlas una por una en la medida de lo posible y describir sus principales características positivas y negativas, haciendo una comparativa personal final para elegir cuál podía ser la mejor opción. Además, hemos intentado hacerlo de una manera sencilla y clara para llegar a todos los tipos de público, tanto los que son avanzados en temas de programación e informática, como los que no.

- *Familiarizarnos con los diferentes entornos de desarrollo asociados, ya sea sobre propios sistemas operativos como Ubuntu 16.04 LTS o entornos virtualizados como Cloudera, además de adentrarnos en la programación con lenguajes desconocidos para nosotros como Scala o R.*

El uso de herramientas y tecnologías Open Source lo hemos asociado con diferentes sistemas operativos y entornos virtualizados, Apache Spark instalado en Ubuntu 16.04 LTS y Apache Hadoop en Cloudera CDH5 que está basado en CentOS. En ambos entornos, hemos tenido que ampliar los conocimientos para utilizarlos de una manera mucho más profunda para poder sacarle el mayor partido. Por otro lado, hemos tenido que aprender y familiarizarnos con nuevos lenguajes de programación como Scala o R, los cuáles tienen un proceso de adaptación y entendimiento, que, en el caso del primer lenguaje citado, ha sido un poco tedioso. El objetivo también ha estado cumplido.

- *En la parte final haremos un proyecto real con una herramienta o tecnología concreta. Dicho proyecto consistirá en predecir las posiciones de diferentes dispositivos conectados a Access Points visibles mediante algoritmos de Machine Learning. Posteriormente se evaluará el potencial que puede tener la herramienta de elección y un proyecto como el nuestro.*

Sinceramente creemos que ha sido la parte más dura del proyecto, ya que al profundizar en la tecnología Apache Spark junto con Machine Learning nos hemos encontrado multitud de dificultades y complicaciones, sobre todo al manipular los datos y aplicarle diferentes algoritmos, que han requerido ayuda y soporte por parte del tutor y mucha investigación en Internet para solventarlo. Aún así, ha habido un gran trabajo detrás de las diferentes comparaciones de algoritmos, para tener unos valores de precisión aceptables y poder extraer conclusiones asociadas. El objetivo se ha cumplido en mayor medida satisfactoriamente.

- *Queremos destacar que, durante la realización del proyecto, otro objetivo no enumerado, pero igual de importante consistía en realizar una buena organización del tiempo y del trabajo, combinándolo con las horas laborales y los estudios (tanto universitarios como extra universitarios de cada uno).*

La correcta gestión del tiempo ha sido un elemento totalmente fundamental al realizar este proyecto, así como cuadrar los horarios y tareas de cada uno para avanzar en su desarrollo.

## 5.2 – Conclusiones generales

Primero de todo, nos gustaría destacar que al inicio de este proyecto teníamos pensado un enfoque diferente al resultante, dado que no sabíamos nada sobre Big Data. Al empezar a realizar estudios sobre este ámbito nos dimos cuenta que realizar la idea inicial era muy complicado, y que nuestros objetivos finales y personales debían ser aprender las utilidades y características que ofrecía Big Data como una primera parte troncal, y una parte secundaria experimental conformaría el desarrollo de estas.

En la parte final de cada capítulo del proyecto, hemos realizado unas conclusiones parciales de cada ámbito, dando a conocer nuestras principales

recomendaciones. Es por eso que, en este apartado, las conclusiones serán de una forma mucho más generalizada en términos de Big Data.

Hemos visto que en el ámbito referente a Big Data hay multitud de conceptos, ideas, aplicaciones y entidades implicadas, y que nos va a acompañar durante muchos años. Esta gran versatilidad hace que tenga muchas aplicaciones y que muchas empresas hagan uso de Big Data diariamente. Nos ha sorprendido el hecho de que tenga tanta importancia hoy en día, ya que, inicialmente no nos imaginábamos que tuviera un impacto tan directo en la sociedad actual.

Haciendo hincapié más concretamente en Machine Learning, creemos que tiene muchísimo potencial como reclamo para las empresas, dado que gracias a sus predicciones se puede, entre otras cosas, establecer patrones de comportamiento para clientes potenciales. Las áreas de Big Data de muchas empresas ya están trabajando hoy en día en algoritmos de Machine Learning, por tanto, es ya un trabajo en auge altamente demandado.

Finalmente, creemos que Big Data es un campo de estudio muy interesante y nos asombra enormemente como, cantidades masivas de datos desordenados (incluso ni relación ni sentido algunos) son transformadas y procesadas para obtener toda clase de resultados y conclusiones.

## 5.3 – Perspectivas de futuro

Big Data es un área muy grande como para exponer todas las herramientas y tecnologías que lo conforman en un mismo proyecto solo, las que hemos presentado aquí conforman una base y un punto de partida para poder profundizar en mayor dimensión. A partir del estudio y desarrollo expuesto, podemos sugerir algunas recomendaciones a modo de ampliación y exploración.

- *Uso y estudio del lenguaje Python en el ecosistema Open Source de Big Data.*

Python es un lenguaje de programación muy versátil y con mucho soporte de librerías, se podría analizar como se desenvuelve en estos entornos.

- *Mayor precisión y menor tiempo en los resultados finales del Capítulo 4.*

El uso de otros algoritmos, un mayor preprocesado de datos o el uso de otros datasets son buenas opciones para obtener mejores resultados.

- *Exploración de las nuevas tendencias Open Source de Big Data, como Apache Storm o Apache Kafka para Streaming.*

Aunque nos hemos centrado en Machine Learning, un ascpecto muy importante de Big Data son las aplicaciones en tiempo real o Streaming, donde están empezando a destacar estos dos ejemplos.

*- Realizar una aplicación móvil del proyecto experimental.*

Tras obtener unos resultados más precisos, el siguiente paso sería realizar una aplicación para móviles que nos pueda indicar la posición de los dispositivos, o incluso incorporando la capacidad de predicción de personas.

*- Profundizar en más detalle en las herramientas y ecosistema de Apache Hadoop.*

Hadoop presenta multitud de herramientas, en este proyecto hemos presentado algunas, pero hay muchas más, las cuáles se podrían analizar en profundidad.

*- Probar algoritmos de Redes Neuronales*

En el apartado de Machine Learning se comentan los tipos de algoritmos que podemos encontrar, uno de ellos es la Red Neuronal, una forma de ampliar el proyecto sería analizando y explorando su uso y sus características.

*- Utilización del lenguaje R en un proyecto experimental.*

Nosotros lo hemos analizado de forma práctica, pero sin realizar ningún diseño propio, para acabar de ver su potencial sería recomendable trabajarlo en más detalle.

*- Probar las diferentes tecnologías en modo distribuido utilizando Microsoft Azure o Amazon Web Services.*

El hecho de no tener que instalar un entorno para usar Big Data, usando alguno de los servicios mencionados, nos permitiría mayor flexibilidad a la hora de realizar aplicaciones.

# CAPÍTULO 6: REFERENCIAS Y BIBLIOGRAFÍA

[1] White T. "*Hadoop: The Definitve Guide – Storage and Analysis at Internet scale*". O'Reilly Media Inc, City of Sebastopol – CA (USA), Última edición Abril del 2015.

[2] Karau H., Konwinski A., Wendell P. y Zaharia M. "*Learning Spark – Lightning-fast data*". O'Reilly Media Inc, City of Sebastopol – CA (USA), Última edición Febrero del 2015.

[3] Ryza S., Laserson U., Owen S., y Wills J. "*Advanced analytics with Spark – Patterns for learning from data at scale*". O'Reilly Media Inc, City of Sebastopol – CA (USA), Última edición Abril del 2015.

[4] Instituto de Ingeniería del Conocimiento. "Curso de Big Data: lo que debes aprender". 3 de Julio de 2016. Disponible online: http://www.iic.uam.es/innovacion/curso-big-data-que-debes-aprender/ [Último acceso: Febrero de 2017]

[5] IBM. "MapReduce: A programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster". 2017. Disponible online: https://www.ibm.com/analytics/us/en/technology/hadoop/mapreduce/#mapreduce-resources [Último acceso: Febrero de 2017]

[6] Cisco Visual Networking Index. "Global Mobile Data Traffic Forecast Update, 2016-2021". 2016. Disponible online: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html [Último acceso: Marzo de 2017]

[7] M. Anderson. "How to Install Hadoop in Stand-Alone Mode on Ubuntu 16.04". 13 de Octubre de 2016. Disponible online: https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-in-stand-alone-mode-on-ubuntu-16-04 [Último acceso: Febrero de 2017]

[8] M. G. Noll. "Running Hadoop on Ubuntu Linux (Single-Node Cluster)". 17 de Junio de 2011. Disponible online: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/ [Último acceso: Febrero de 2017]

[9] G. Raha. "How to Install Apache Spark on Ubuntu 16.04 / Debian 8". 4 de Junio de 2017. Disponible online: https://www.techbrown.com/install-apache-spark-ubuntu-16-04-debian-8.shtml [Último acceso: Febrero de 2017]

[10] Cloudera. "QuickStarts for CDH 5.12". 2017. Disponible online: https://www.cloudera.com/downloads/quickstart_vms/5-12.html [Último acceso: Marzo de 2017]

[11] Cran-R Project. “Ubuntu Packages for R”. 3 de Mayo de 2017. Disponible online: https://cran.r-project.org/bin/linux/ubuntu/README [Último acceso: Marzo de 2017]

[12] The Apache Software Foundation y otros desarrolladores. “Installation Sparklyr, Connecting to Spark, Using dplyr, Machine Learning”. 2017. Disponible online: https://spark.rstudio.com/index.html [Último acceso: Mayo de 2017]

[13] The Apache Software Foundation y otros desarrolladores. “Spark Machine Learning Library (MLlib) in R”. 2017. Disponible online: https://spark.rstudio.com/articles/guides-mllib.html [Último acceso: Junio de 2017]

[14] The Apache Software Foundation y otros desarrolladores. “Sparkling Water (H2O) Machine Learning”. 2017. Disponible online: https://spark.rstudio.com/articles/guides-h2o.html [Último acceso: Junio de 2017]

[15] T. McGrath. “Spark Streaming Example – How to Stream from Slack”. 29 de Agosto de 2016. Disponible online: https://www.supergloo.com/fieldnotes/spark-streaming-example-from-slack/ [Último acceso: Junio de 2017]

[16] C. McDonald. “How to get started using Apache Spark GraphX with Scala”. 8 de Marzo de 2016. Disponible online: https://mapr.com/blog/how-get-started-using-apache-spark-graphx-scala/ [Último acceso: Junio de 2017]

[17] Apache Spark. “Spark programming guide”. 2017. Disponible online: http://spark.apache.org/docs/0.8.0/scala-programming-guide.html#parallelized-collections [Último acceso: Agosto de 2017]

[18] Apache Spark. “Spark Conf”. 2017. Disponible online: https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.SparkConf [Último acceso: Agosto de 2017]

[19] Stack Overflow. “Stack Overflow Business Solutions: Looking to understand, engage, or hire developers”. 2017. Disponible online: https://stackoverflow.com/ [Último acceso: Agosto de 2017]

[20] GitHub Inc.. “Built for developers”. 2017. Disponible online: https://github.com/ [Último acceso: Agosto de 2017]

[21] A. Krot. “Social Network Analysis. Spark GraphX”. 26 de Marzo de 2015. Disponible online: https://kukuruku.co/post/social-network-analysis-spark-graphx/ [Último acceso: Junio de 2017]

[22] Gerardnico. “Machine Learning - (Baseline|Naive) classification (Zero R)”. 19 de Marzo de 2014. Disponible online: https://gerardnico.com/wiki/data_mining/baseline [Último acceso: Junio de 2017]

[23] Databricks. “Machine Learning”. 2017. Disponible online: https://databricks.com/product/getting-started-guide/machine-learning [Último acceso: Julio de 2017]

[24] X. Meng y otros colaboradores. “MLlib: Scalable Machine Learning on Spark”. 2017. Disponible online: https://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf [Último acceso: Junio de 2017]

[25] IndoorLoc Platform - UJI. “Datasets”. 13 de Octubre de 2015. Disponible online: http://indoorlocplatform.uji.es/databases/all/ [Último acceso: Agosto de 2017]

[26] A. Moreira. M.J. Nicolau. F. Meneses. A. Costa. “Wi-Fi Fingerprinting in the Real World RTLS@UM at the EvAAL Competition”. *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Disponible online: http://ieeexplore.ieee.org/document/7346967/ [Último acceso: Agosto de 2017]

[27] Apache Spark. “Apache Spark: Lightning-fast cluster computing”. 11 de Julio de 2017. Disponible online: http://spark.apache.org/downloads.html [Último acceso: Febrero de 2017]

[28] Apache Hadoop. “Apache Hadoop Releases”. 7 de Julio de 2017. Disponible online: http://hadoop.apache.org/releases.html [Último acceso: Febrero de 2017]

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# ANEXOS

**TÍTULO DEL TFG: "Análisis, uso y desarrollo experimental de herramientas y tecnologías Open Source en Big Data"**

**TITULACIÓN: Grado en Ingeniería Telemática y Grado en Sistemas de Telecomunicación**

**AUTORES: Jean-Paul Fannes Claverol**

**Marc Pallejà Mairena**

**DIRECTOR: Roc Meseguer Pallarès**

**FECHA: 12 de Septiembre del 2017**

# ANEXO I. LENGUAJE R - INSTALACIÓN DEL ENTORNO Y USO

La instalación es sencilla para cualquier usuario familiarizado con el entorno y la línea de comandos asociada a distribuciones basadas en Debian en este caso, hace falta instalar R desde los repositorios de CRAN (https://cran.r-project.org/bin/linux/ubuntu).
Para ello, lo primero que tendremos que hacer será editar el fichero *sources.list*, ya sea por consola con los comandos “nano” o “vi” o bien con un editor de texto gráfico como por ejemplo “gedit”, y añadir la URL de CRAN que se identifica con nuestra versión de Ubuntu, en nuestro caso, Xenial (16.04), tal y como muestra la siguiente Figura A.1

```
deb http://security.ubuntu.com/ubuntu xenial-security main restricted
# deb-src http://security.ubuntu.com/ubuntu xenial-security main restricted
deb http://security.ubuntu.com/ubuntu xenial-security universe
# deb-src http://security.ubuntu.com/ubuntu xenial-security universe
deb http://security.ubuntu.com/ubuntu xenial-security multiverse
# deb-src http://security.ubuntu.com/ubuntu xenial-security multiverse
deb https://cran.rediris.es/bin/linux/ubuntu xenial/
```

**Figura A.1** – Edición del archivo sources.list y URL añadida de CRAN para instalar R

Cabe destacar que los archivos de Ubuntu con CRAN tienen que estar signados con una clave para que se pueda instalar el repositorio, sino no nos dejará proceder a la instalación. Para signarlos, utilizaremos la clave de Michael Rutter (E084DAB9), y para hacerlo haremos uso del siguiente comando (Figura A.2)

```
marc@ubuntu:~$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9
Executing: /tmp/tmp.8eW59YJvET/gpg.1.sh --keyserver
keyserver.ubuntu.com
--recv-keys
E084DAB9
gpg: requesting key E084DAB9 from hkp server keyserver.ubuntu.com
gpg: key E084DAB9: public key "Michael Rutter <marutter@gmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

**Figura A.2** – Comando para signar el repositorio de R para su posterior instalación

Todo seguido podemos proceder a la instalación de R, primero actualizando los repositorios y luego haciendo su posterior instalación (Figura A.3)

```
marc@ubuntu:~$ sudo apt-get update
marc@ubuntu:~$ sudo apt-get install r-base
```

**Figura A.3** – Actualización de repositorios y posterior instalación de R

Una vez instalado, podemos ejecutar R por consola para comprobar su correcto funcionamiento, tal y como muestra la Figura A.4.

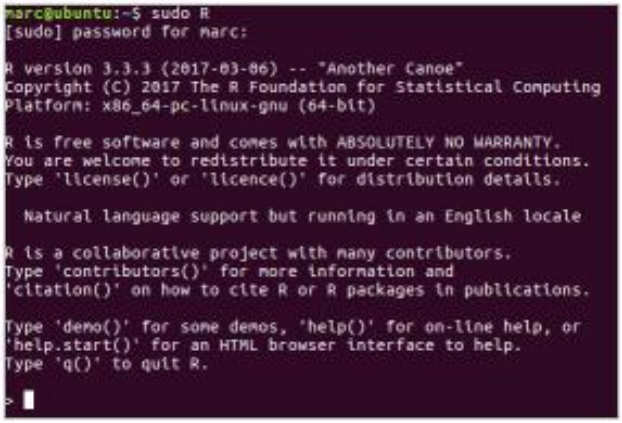



**Figura A.4** – Uso de R mediante consola para comprobar su funcionamiento

Aunque podamos utilizar R mediante consola, es mucho más cómodo utilizar un IDE, en este caso utilizaremos RStudio. Para ello nos lo descargaremos de la página oficial https://www.rstudio.com/, seleccionaremos la versión de escritorio gratuita y seleccionaremos el instalador apropiado dependiendo de nuestro sistema operativo, en nuestro caso utilizamos Ubuntu 16.04 LTS versión de 64 bits. Cabe destacar que después de descargarnos la versión apropiada, utilizaremos el comando “sudo dpkg -i [ruta donde se encuentra el instalador descargado]” para instalarlo.

Una vez instalado, ya podremos abrir el programar y veremos que la pantalla principal es como la Figura A.5

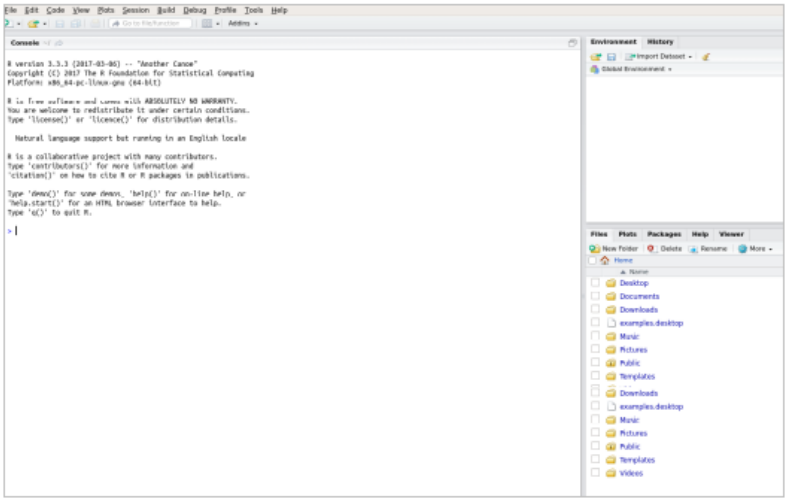


**Figura A.5** – Uso de R mediante la interfas de RStudio

Como primera toma de contacto con RStudio vemos que hay una zona dedicada a cada parte, la parte izquierda-central está dedicada a la consola de R y a los archivos que podamos abrir en ese formato y la parte superior derecha se centra en el historial y variables y la parte inferior derecha se centra en los paquetes y gráficos. Para hacer diferentes programas en R, veremos que harán falta muchos paquetes asociados previa descarga, por tanto, instalaremos los que nos hagan falta en cualquier momento.

# ANEXO II. INSTALACIÓN Y USO DE APACHE HADOOP EN MODO ‘SINGLE NODE’

Hemos realizado la instalación de Apache Hadoop en modo Single Node en Ubuntu 16.04 LTS.

El primer paso que tendremos que realizar consiste en asegurarnos que tenemos la última versión instalada de JDK en nuestro sistema, sino es así deberemos instalarla obligatoriamente para que todo funcione correctamente. Tal y como muestra la siguiente Figura A.6:

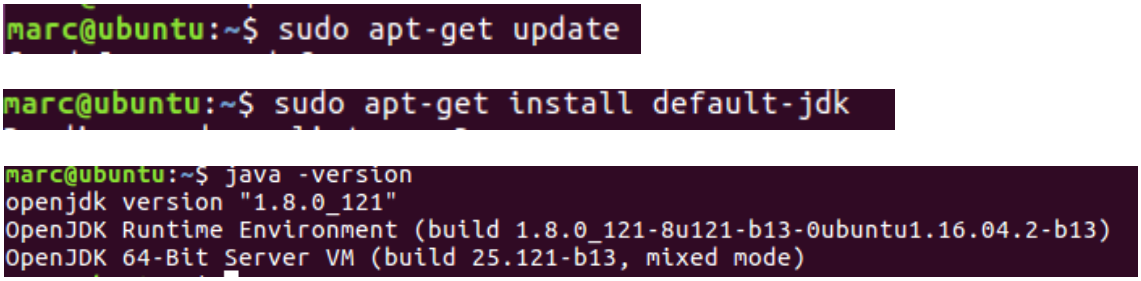



**Figura A.6** – Instalación de JDK y posterior comprobación de la versión instalada

El segundo paso es descargarnos la versión estable de Hadoop que más nos interese, para ello iremos a la página oficial y nos descargaremos dicha versión. En nuestro caso, elegiremos la versión 2.7.3 en formato tar.gz. Para descomprimirla y ejecutarla después.

Una vez tengamos la versión descargada, el tercer paso consistirá en bajarse el archivo tar.gz.mds para poder firmar con una clave el primer archivo y poder ejecutarlo. Tal y como muestra la Figura A.7:

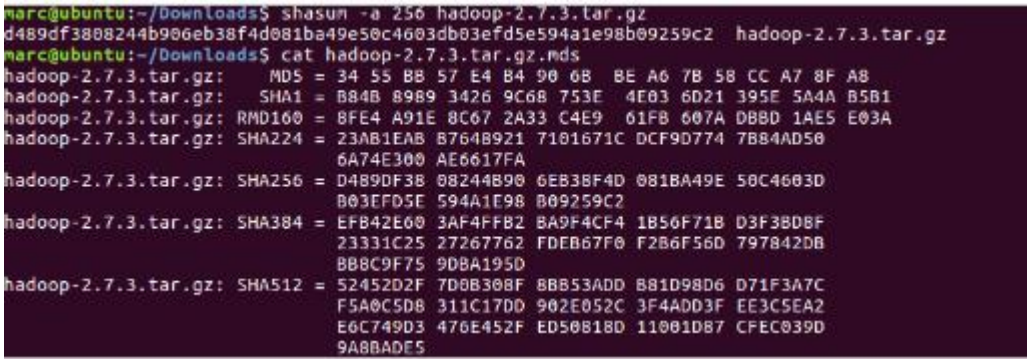



**Figura A.7** – Firma del archivo tar.gz con la clave correspondiente

El próximo paso será descomprimir el archivo tar.gz en el directorio que queremos que se encuentre Hadoop (Figura A.8)

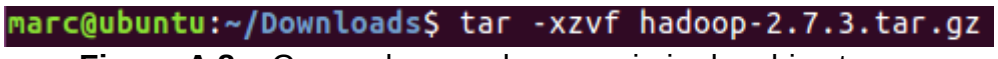



**Figura A.8** – Comando para descomprimir el archivo tar.gz

Una vez descomprimido todo, el siguiente paso será abrir el archivo 'hadoop-env.sh' para añadirle e indicarle la ruta de acceso del JDK para que se pueda ejecutar corretamente y sin problemas, tal y como muestra la Figura A.9.

```
marc@ubuntu:~/Downloads$ readlink -f /usr/bin/java | sed "s:bin/java::"
/usr/lib/jvm/java-8-openjdk-amd64/jre/
marc@ubuntu:~/Downloads$
marc@ubuntu:~/Downloads$
marc@ubuntu:~/Downloads$
marc@ubuntu:~/Downloads$ sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

```
# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/
```

**Figura A.9** – Ruta de acceso al JDK añadida al archivo 'hadoop-env.sh'

Una vez realizado este paso, podemos ejecutar ya Hadoop sin ningún problema y comprobar que funciona correctamente. En nuestro caso, cogeremos un archivo .xml de la misma carpeta de Hadoop y ejecutaremos un proceso de MapReduce.

Para realizar dicho ejemplo, crearemos una carpeta input donde pondremos el archivo .xml, y una vez realizado, ejecutaremos el comando de Hadoop con el .jar que viene ya en la carpeta de ejemplo para poder ejecutar el proceso MapReduce, tal y como se muestra en la Figura A.10.

```
marc@ubuntu:/usr/local/hadoop$ sudo mkdir ~/input
[sudo] password for marc:
```

```
17/03/22 19:56:42 INFO mapreduce.Job: Job job_local2066411160_0001 failed with state FAILED due to: NA
17/03/22 19:56:42 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=2562413
                FILE: Number of bytes written=4649359
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=745
                Map output records=7
                Map output bytes=127
                Map output materialized bytes=89
                Input split bytes=813
                Combine input records=7
                Combine output records=2
                Reduce input groups=0
                Reduce shuffle bytes=89
                Reduce input records=0
                Reduce output records=0
                Spilled Records=2
                Shuffled Maps =8
                Failed Shuffles=0
                Merged Map outputs=8
                GC time elapsed (ms)=354
                Total committed heap usage (bytes)=1261010944
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=26007
        File Output Format Counters
                Bytes Written=0
17/03/22 19:56:42 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
17/03/22 19:56:42 INFO mapreduce.JobSubmitter: Cleaning up the staging area file:/tmp/hadoop-marc/mapred/staging/marc1594776311/.staging/job_local1594776311_0002
```

**Figura A.10** – Creación de la carpeta input y ejecución del proceso MapReduce

Una vez finalizado, se habrá creado una carpeta output con el resultado.

La ejecución se produce toda en local, sin necesidad de utilizar el proceso de SSH para ejecutarse en diferentes nodos.

# ANEXO III. UTILIZACIÓN DE WORDCOUNT EN APACHE HADOOP

Tal y como hemos comentado en el Capítulo 3 con la finalidad de analizar y comparar las tecnologías, hemos utilizado el ejemplo de WordCount tanto en Hadoop como en Spark para ver las diferentes características de cada uno.

En esta parte del Anexo, adjuntaremos el código para ejecutar el proceso de MapReduce en Java y en el entorno de Eclipse que viene instalado por defecto en Cloudera.

Para ayudarnos en este proceso, hemos instalado una herramienta externa en el entorno de Eclipse añadiendo el .jar en la parte de herramientas, que nos permitirá crear clases propias de Map y Reduce, con tal de agilizar todo el proceso para que sea más cómodo. Así pues, primero crearemos una clase Map tal y como muestra la siguiente Figura A.11

```java
package hadoop.marc;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.*;

//import org.apache.hadoop.io.Writable;
//import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class mappper extends MapReduceBase implements Mapper <LongWritable, Text, Text, IntWritable>  {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();
      public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           output.collect(word, one);
        }
      }
    }
```

**Figura A.11** – Clase con la función Map en Java

Todo seguido crearemos la clase Reduce tal y como muestra la Figura A.12

```java
package hadoop.marc;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

**Figura A.12** – Clase con la función Reduce en Java

Finalmente, una vez creadas las clases Map y Reduce, crearemos la función principal WordCount, donde se encontrará el main o programa principal con tal

de llamar a las dos clases anteriormente mencionadas para ejecutarlas, indicar como leer los textos y en que carpetas están localizados. La clase principal se muestra en la Figura A.13

```
13 public class WordCount {
14
15     public static void main(String[] args) {
16         JobClient client = new JobClient();
17         JobConf conf = new JobConf(hadoop.marc.WordCount.class);
18         conf.setJobName("wordcount");
19
20         //  specify output types
21         conf.setOutputKeyClass(Text.class);
22         conf.setOutputValueClass(IntWritable.class);
23
24         //  specify input and output Format and DIRECTORIES (not files)
25         conf.setInputFormat(TextInputFormat.class);
26         conf.setOutputFormat(TextOutputFormat.class);
27
28         FileInputFormat.setInputPaths(conf, new Path("input"));
29         FileOutputFormat.setOutputPath(conf, new Path("output"));
30
31         conf.setMapperClass(hadoop.marc.mappper.class);
32                     conf.setCombinerClass(hadoop.marc.WCReducer.class);
33         conf.setReducerClass(hadoop.marc.WCReducer.class);
34
35         client.setConf(conf);
36         try {
37             JobClient.runJob(conf);
38         } catch (Exception e) {
39             e.printStackTrace();
40         }
41     }
42
43 }
```

**Figura A.13** – Clase principal para ejecutar el proceso de WordCount en Java

Una vez nos aseguremos que las 3 clases no presentan ningún tipo de error y que están todos los imports adecuados, podremos proceder a crear el .jar para ejecutar el proceso de MapReduce desde la consola de Hadoop.

# ANEXO IV. EJEMPLO EN R – COMPARACIÓN DE LAS FUNCIONES DE MACHINE LEARNING

En esta parte del Anexo, adjuntaremos el código íntegro del ejemplo de la parte de R de Machine Learning. Los comentarios explicativos están en el propio código, con tal de agilizar la explicación de una manera rápida y clara.

```
library(sparklyr)
library(dplyr)
library(tidyr)
library(titanic)
library(ggplot2)
library(purrr)

# Conectamos con el clúster local de Spark y cargamos los datos
sc <- spark_connect(master = "local")
spark_read_parquet(sc, name = "titanic", path = "titanic-parquet")
titanic_tbl <- tbl(sc, "titanic")

# Transformamos los datos con la API Spark SQL
titanic2_tbl <- titanic_tbl %>%
  mutate(Family_Size = SibSp + Parch + 1L) %>%
  mutate(Pclass = as.character(Pclass)) %>%
  filter(!is.na(Embarked)) %>%
  mutate(Age = if_else(is.na(Age), mean(Age), Age)) %>%
  sdf_register("titanic2")

# Transformamos el tamaño de los datos con la API Spark ML
titanic_final_tbl <- titanic2_tbl %>%
  mutate(Family_Size = as.numeric(Family_size)) %>%
  sdf_mutate(
    Family_Sizes = ft_bucketizer(Family_Size, splits = c(1,2,5,12))
    ) %>%
  mutate(Family_Sizes = as.character(as.integer(Family_Sizes))) %>%
  sdf_register("titanic_final")

# Partimos los datos para poder trabajar más cómodamente
partition <- titanic_final_tbl %>%
  mutate(Survived = as.numeric(Survived), SibSp = as.numeric(SibSp), Parch = as.numeric(Parch)) %>%
  select(Survived, Pclass, Sex, Age, SibSp, Parch, Fare, Embarked, Family_Sizes) %>%
  sdf_partition(train = 0.75, test = 0.25, seed = 8585)

# Creamos las tablas de referencia, más concretamente 2 "train" y "test"
train_tbl <- partition$train
test_tbl <- partition$test

# Modelo de supervivencia en función de varios predictores
ml_formula <- formula(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked + Family_Sizes)

# Modelo de Regresión Logística
(ml_log <- ml_logistic_regression(train_tbl, ml_formula))

## Modelo de Árbol de decisión (Decision Tree)
ml_dt <- ml_decision_tree(train_tbl, ml_formula)

## Modelo de Bosque aleatorio (Random Forest)
ml_rf <- ml_random_forest(train_tbl, ml_formula)

## Gradient Boosted Tree Model
ml_gbt <- ml_gradient_boosted_trees(train_tbl, ml_formula)

## Naive Bayes Model
ml_nb <- ml_naive_bayes(train_tbl, ml_formula)

## Modelo de Red neuronal (Neural Network)
ml_nn <- ml_multilayer_perceptron(train_tbl, ml_formula, layers = c(11,15,2))

# Agrupación de los diferentes modelos de ML en una lista
ml_models <- list(
  "Logistic" = ml_log,
  "Decision Tree" = ml_dt,
  "Random Forest" = ml_rf,
  "Gradient Boosted Trees" = ml_gbt,
  "Naive Bayes" = ml_nb,
  "Neural Net" = ml_nn
)
```

```
# Creamos una función para apuntar los resultados
score_test_data <- function(model, data=test_tbl){
  pred <- sdf_predict(model, data)
  select(pred, Survived, prediction)
}

# Apuntamos todos los modelos de ML elegidos
ml_score <- lapply(ml_models, score_test_data)

# Función lift
calculate_lift <- function(scored_data) {
  scored_data %>%
    mutate(bin = ntile(desc(prediction), 10)) %>%
    group_by(bin) %>%
    summarize(count = sum(Survived)) %>%
    mutate(prop = count / sum(count)) %>%
    arrange(bin) %>%
    mutate(prop = cumsum(prop)) %>%
    select(-count) %>%
    collect() %>%
    as.data.frame()
}

# Inicializamos los resultados
ml_gains <- data.frame(bin = 1:10, prop = seq(0, 1, len = 10), model = "Base")

# Calculamos la función lift
for(i in names(ml_score)){
  ml_gains <- ml_score[[i]] %>%
    calculate_lift %>%
    mutate(model = i) %>%
    rbind(ml_gains, .)
}

# Mostramos los resultados en una gráfica
ggplot(ml_gains, aes(x = bin, y = prop, colour = model)) +
  geom_point() + geom_line() +
  ggtitle("Lift Chart for Predicting Survival - Test Data Set") +
  xlab("") + ylab("")

# Función para calcular la precisión
calc_accuracy <- function(data, cutpoint = 0.5){
  data %>%
    mutate(prediction = if_else(prediction > cutpoint, 1.0, 0.0)) %>%
    ml_classification_eval("prediction", "Survived", "accuracy")
}

# Calculamos la precisión y el AUC
perf_metrics <- data.frame(
  model = names(ml_score),
  AUC = 100 * sapply(ml_score, ml_binary_classification_eval, "Survived", "prediction"),
  Accuracy = 100 * sapply(ml_score, calc_accuracy),
  row.names = NULL, stringsAsFactors = FALSE)
# Mostramos los resultados en una gráfica
gather(perf_metrics, metric, value, AUC, Accuracy) %>%
  ggplot(aes(reorder(model, value), value, fill = metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  xlab("") +
  ylab("Percent") +
  ggtitle("Performance Metrics")
```

**Figura A.14** – Código íntegro del ejemplo dónde se comparan diferentes algoritmos de Machine Learning

# ANEXO V. OTROS EJEMPLOS DE MACHINE LEARNING EN R

En esta parte del Anexo, adjuntaremos otros ejemplos que nos han parecido interesantes de Machine Learning en R.

*Ejemplo 1 - Uso de K-Means Clustering*

El uso de la función K-Means Clustering nos permite dividir un conjunto de datos en grupos. K-Means se refiere a la agrupación de puntos en k grupos, haciendo que la suma de sus cuadrados al centro del clúster asignado se minimicen. Utilizaremos unos datos que miden los atributos de 150 flores con 3 especies diferentes de iris. Principalmente contaremos con 3 clústers con 2 principales atributos: Ancho y Largo, donde haremos una predicción sobre ellos y las 3 especies de iris. Aquí solo mostraremos el resultado final que nos proporciona esta función, que es el siguiente:


**Gráfico A.15** – Representación de K-Means Clustering

Podemos observar que hay 3 clústers diferenciados correspondiéndose con cada especie de iris, y cada uno se ajusta más o menos a la predicción de Ancho (eje vertical) y Largo (eje horizontal).

*Ejemplo 2 - Uso de Regresión Lineal*

El uso de la Regresión Lineal permite modelar la relación lineal entre una variable de respuesta y una o más variables explicativas.

En el siguiente ejemplo intentaremos predecir el consumo de combustible basándonos en el peso y el número de cilindros que contiene. Primero, empezaremos cargando las librerías, conectando con Spark y copiando los datos localmente a Spark.

La segunda parte del código es la siguiente (donde extraeremos una parte comentada y explicada por nosotros):

```
# Transformamos los datos con Spark SQL y con otras funciones
# Quitamos los cocches con menos de 100 CV
# Ponemos los coches en dos grupos basandonos en sus cilindros
# Utilizamos funciones de Spark para separar los datos en test y training
partitions <- mtcars_tbl %>%
  filter(hp >= 100) %>%
  sdf_mutate(cyl8 = ft_bucketizer(cyl, c(0,8,12))) %>%
  sdf_partition(training = 0.5, test = 0.5, seed = 888)

# Ajustamos un modelo lineal mediante Spark ML. El modelo "mpg" va en función del peso y de los cilindros
fit <- partitions$training %>%
  ml_linear_regression(mpg ~ wt + cyl)



# Utilizamos el ajuste de modelo de Spark para predecir el consumo medio de combustible en nuestros datos de prueba
# Y comparamos la respuesta prevista con el verdadero consumo de combustible medido.
pred <- sdf_predict(fit, partitions$test) %>%
  collect
```

**Figura A.16** – Segunda parte del código explicada

En esta parte del código hemos combinado funciones y algoritmos, transformadores de datos, utilidades y otras extensiones para acabar de complementar el ejemplo.

Finalmente, hacemos el gráfico de la predicción contra el consumo real:

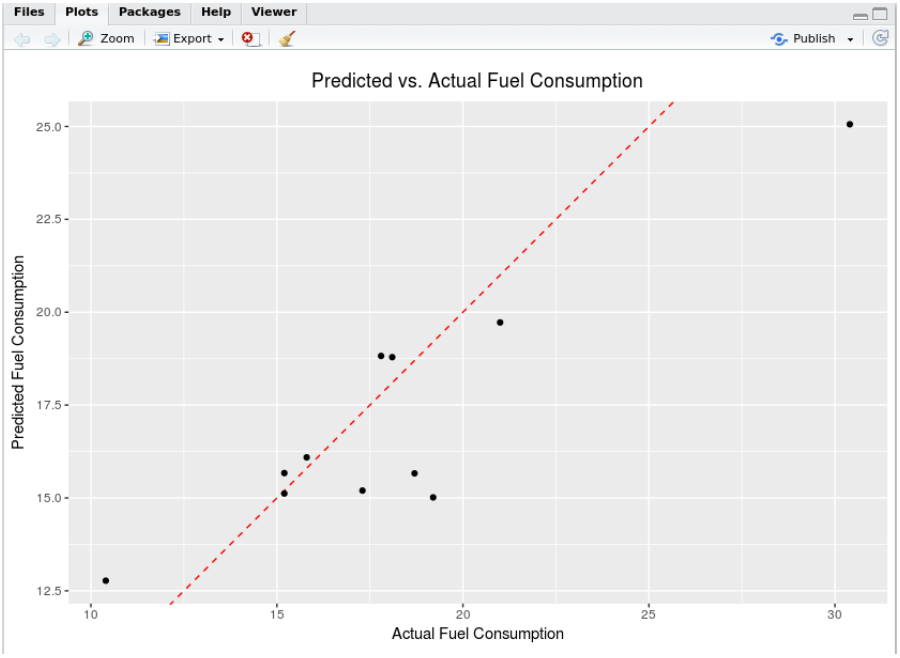



**Gráfico A.17** – Representación de Regresión Lineal donde observamos la diferencia entre la predicción y el valor real de consumo

Si comparamos la parte izquierda que corresponde a la predicción de combustible frente a la parte derecha que corresponde al actual consumo de combustible. A grandes rasgos, podemos comprobar que la predicción hace una gran aproximación a los valores reales.

*Ejemplo 3 - Uso de Análisis de Componentes Principales (PCA)*

Si utilizamos el Análisis de Componentes Principales (Principal Components Analysis – PCA) realizamos una reducción dimensional, es decir, es un método estadístico para encontrar una rotación tal que la primera coordenada tiene la mayor varianza posible y cada coordenada sucesiva a su vez tiene la mayor varianza posible. Principalmente sirve para reducir el tamaño de los datos que están en forma de matriz basándose en el cálculo de autovalores y autovectores y procesando los datos importantes y quitando los otros.

Utilizando los datos del ejemplo 1 de este Anexo, y haciendo uso de la función "*ml_pca*" podemos observar el siguiente resultado:

```
Explained variance:
[not available in this version of Spark]

Rotation:
                    PC1         PC2         PC3        PC4
Sepal_Length -0.36138659 -0.65658877  0.58202985  0.3154872
Sepal_Width   0.08452251 -0.73016143 -0.59791083 -0.3197231
Petal_Length -0.85667061  0.17337266 -0.07623608 -0.4798390
Petal_Width  -0.35828920  0.07548102 -0.54583143  0.7536574
```

**Tabla A.18** – Atributos de rotación para cada atributo

Dónde podemos observar diferentes resultados de la rotación para cada atributo de los datos.

# ANEXO VI. EJEMPLO DEL ARCHIVO BUILD.SBT Y PASOS PARA EJECUTAR SBT Y CREAR EL .JAR

En la parte del trabajo de Spark hemos usado la herramienta SBT para importar librerías y hacer los jars (como alternativa a Maven). En cada nuevo proyecto realizado con IntelIJ IDEA hace falta un archivo build.sbt que será el que dictaminará que librerías estamos usando, ya que Spark presenta algunas de muy interesantes, será imprescindible el correcto uso de este archivo.

```
name := "grafics"
version := "1.0"
scalaVersion := "2.11.6"
libraryDependencies += "org.apache.spark" % "spark-core_2.10" % "1.0.2"
libraryDependencies += "org.apache.spark" % "spark-sql_2.10" % "1.0.2"
libraryDependencies += "org.apache.spark" % "spark-graphx_2.10" % "1.0.2"
```

**Figura A.19** – Creación del archivo build.sbt

Como vemos en la Figura A.19, tenemos unes líneas puramente informativas, el nombre y la versión de nuestro proyecto; seguido de la versión de Scala, que recomendamos que sea la 2.11.6 por compatibilidad con la última versión de Spark y finalmente las librerías que usaremos. Aunque con el tiempo se mecaniza el proceso de escribir las librerías no es trivial al principio, ya que hay que cerciorarse que se referencia correctamente la versión de la librería que tenemos en nuestra versión de Spark local.

Una vez tenemos el proyecto listo, lo que tendremos que hacer es crear el .jar, para hacerlo tendremos que ir a la carpeta personal donde se encuentra dicho proyecto y poner el siguiente comando:

***sbt clean***

Una vez se haya ejecutado correctamente, ejecutaremos el siguiente comando:

***sbt package***

Con estos dos comandos crearemos el archivo .jar para ser ejecutado. Finalmente, y dependiendo del proyecto que queramos ejecutar, podemos crear una carpeta input para poner algún tipo de fichero de texto que queramos leer e iremos a la carpeta donde tenemos Apache Spark corriendo en local, más concretamente en la carpeta bin para poder ejecutarlo correctamente. Un ejemplo de ejecución del archivo .jar en Spark podría ser este:

***sudo ./spark-submit --class WordCount --master local[*] /root/IdeaProjects/Prova/target/scala-2.11/prova_2.11-1.0.jar /home/marc/input /home/marc/output***

Donde indicaremos las clase principal del programa y el directorio correspondiente al proyecto. Las carpetas input y output son opcionales.

# ANEXO VII. EJEMPLO COMPLETO CON APACHE SPARK, LIBRERÍA SQL/GRAPHX

A continuación, presentaremos el código completo usado para el ejemplo de uso de la librería de SQL/GraphX.
Para poner en contexto este código se debería visitar la página web (que se puede encontrar en el Capítulo 6: Referencias y Bibliografía) donde encontraremos más explicaciones de lo que hace cada segmento.

```
package solutions

import org.apache.spark._

import org.apache.spark.rdd.RDD
import org.apache.spark.util.IntParam
import org.apache.spark.sql.SQLContext

import org.apache.spark.graphx._
import org.apache.spark.graphx.util.GraphGenerators
import org.apache.spark.util.StatCounter

case class Flight(dofM: String, dofW: String, carrier: String, tailnum: String, flnum: Int, org_id: Long,
origin: String, dest_id: Long, dest: String, crsdeptime: Double, deptime: Double, depdelaymins: Double,
crsarrtime: Double, arrtime: Double, arrdelay: Double, crselapsedtime: Double, dist: Int)

object FlightApp {

  def parseFlight(str: String): Flight = {
    val line = str.split(",")
    Flight(line(0), line(1), line(2), line(3), line(4).toInt, line(5).toLong,
      line(6), line(7).toLong, line(8), line(9).toDouble, line(10).toDouble,
      line(11).toDouble, line(12).toDouble, line(13).toDouble, line(14).toDouble, line(15).toDouble, line
(16).toInt)
  }

  def main(args: Array[String]) {
  val conf = new SparkConf().setAppName("SparkDFebay")
  val sc = new SparkContext(conf)
  // load the data into a RDD
  val textRDD = sc.textFile("/home/jpf/Downloads/rita2014jan.csv")


 // parse the RDD of csv lines into an RDD of flight classes
 val flightsRDD = textRDD.map(parseFlight).cache()

 // create airports RDD with ID and Name
 val airports = flightsRDD.map(flight => (flight.org_id, flight.origin)).distinct
 airports.take(1)

 // Defining a default vertex called nowhere
 val nowhere = "nowhere"

 // create routes RDD with srcid, destid, distance
 val routes = flightsRDD.map(flight => ((flight.org_id, flight.dest_id), flight.dist)).distinct

 routes.cache


 // AirportID is numerical - Mapping airport ID to the 3-letter code
 val airportMap = airports.map { case ((org_id), name) => (org_id -> name) }.collect.toList.toMap

 //airportMap: scala.collection.immutable.Map[Long,String] = Map(13024 -> LMT, 10785 -> BTV, 14574 -> ROA,
14057 -> PDX, 13933 -> ORH, 11898 -> GFK, 14709 -> SCC, 15380 -> TVC,

 // Defining the routes as edges
 val edges = routes.map { case ((org_id, dest_id), distance) => Edge(org_id.toLong, dest_id.toLong,
```

```
distance) }

  //Defining the Graph
  val graph = Graph(airports, edges, nowhere)


  // LNumber of airports
  val numairports = graph.numVertices

  // graph vertices
  graph.vertices.take(2)

  // graph edges
  graph.edges.take(2)

  // routes > 1000 miles distance?
  graph.edges.filter { case (Edge(org_id, dest_id, distance)) => distance > 1000 }.take(3)
  // res9: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(10140,10397,1269), Edge(10140,10821,1670),
Edge(10140,12264,1628))

  // Number of routes
  val numroutes = graph.numEdges

  // The EdgeTriplet class extends the Edge class by adding the srcAttr and dstAttr members which contain
the source and destination properties respectively.
  graph.triplets.take(3).foreach(println)

  // Define a reduce operation to compute the highest degree vertex
  def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
    if (a._2 > b._2) a else b
  }

// Compute the max degrees
  val maxInDegree: (VertexId, Int) = graph.inDegrees.reduce(max)
  // maxInDegree: (org.apache.spark.graphx.VertexId, Int) = (10397,152)
  val maxOutDegree: (VertexId, Int) = graph.outDegrees.reduce(max)
  // maxOutDegree: (org.apache.spark.graphx.VertexId, Int) = (10397,153)
  val maxDegrees: (VertexId, Int) = graph.degrees.reduce(max)
  // maxDegrees: (org.apache.spark.graphx.VertexId, Int) = (10397,305)

  // we can compute the in-degree of each vertex (defined in GraphOps) by the following:
  // which airport has the most incoming flights?
  graph.inDegrees.collect.sortWith(_._2 > _._2).map(x => (airportMap(x._1), x._2))
  //res46: Array[(String, Int)] = Array((ATL,152), (ORD,145), (DFW,143), (DEN,132), (IAH,107), (MSP,96),
(LAX,82), (EWR,82), (DTW,81), (SLC,80),
 // graph.outDegrees.join(airports).sortBy(_._2._1, ascending = false).take(1)
    //graph.outDegrees.innerJoin(airports).sortBy(_._2._1, ascending = false).take(1)
  //val maxout = graph.outDegrees.join(airports).sortBy(_._2._1, ascending = false).take(3)
  //res46: Array[(String, Int)] = Array((ATL,152), (ORD,145), (DFW,143), (DEN,132), (IAH,107), (MSP,96),
(LAX,82), (EWR,82), (DTW,81), (SLC,80),
  val maxIncoming = graph.inDegrees.collect.sortWith(_._2 > _._2).map(x => (airportMap(x._1), x._2)).take(3)
  maxIncoming.foreach(println)


  //  maxout.foreach(println)

  val maxOutgoing = graph.outDegrees.collect.sortWith(_._2 > _._2).map(x => (airportMap(x._1), x._2)).take(3)
  maxOutgoing.foreach(println)

  // What are the top 10 flights from airport to airport?
  //graph.triplets.sortBy(_.attr, ascending = false).map(triplet =>
  //  "There were " + triplet.attr.toString + " flights from " + triplet.srcAttr + " to " + triplet.dstAttr
 ".").take(10)

  val sourceId: VertexId = 13024
  // 50 + distance / 20
  graph.edges.filter { case (Edge(org_id, dest_id, distance)) => distance > 1000 }.take(3)

  val gg = graph.mapEdges(e => 50.toDouble + e.attr.toDouble / 20)
  val initialGraph = gg.mapVertices((id, _) => if (id == sourceId) 0.0 else Double.PositiveInfinity)
  }
```

**Figura A.20** – Código completo del ejemplo de SQL/GraphX

# ANEXO VIII. EJEMPLO COMPLETO DE APACHE SPARK, LIBRERÍA SPARK STREAMING

El código de Spark Streaming se divide en dos partes, la aplicación principal y una clase, el receptor.

```
*/
package com.supergloo

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}

/**
  * Spark Streaming Example App
  */
object SlackStreamingApp {

  def main(args: Array[String]) {
    val conf = new SparkConf().setMaster(args(0)).setAppName("SlackStreaming")
    val ssc = new StreamingContext(conf, Seconds(5))
    val stream = ssc.receiverStream(new SlackReceiver(args(1)))
    stream.print()
    if (args.length > 2) {
      stream.saveAsTextFiles(args(2))
    }
    ssc.start()
    ssc.awaitTermination()
  }
```

**Figura A.21** – Código de la aplicación principal “Main” - Ejemplo de Streaming

```
package com.supergloo

import org.apache.spark.Logging
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.receiver.Receiver
import org.jfarcand.wcs.{TextListener, WebSocket}

import scala.util.parsing.json.JSON
import scalaj.http.Http

/**
  * Spark Streaming Example Slack Receiver from Slack
  */
class SlackReceiver(token: String) extends Receiver[String](StorageLevel.MEMORY_ONLY) with Runnable with
Logging {

  private val slackUrl = "https://slack.com/api/rtm.start"

  @transient
  private var thread: Thread = _

  override def onStart(): Unit = {
    thread = new Thread(this)
    thread.start()
  }

  override def onStop(): Unit = {
    thread.interrupt()
  }

  override def run(): Unit = {
    receive()
  }

  private def receive(): Unit = {
    val webSocket = WebSocket().open(webSocketUrl())
    webSocket.listener(new TextListener {
      override def onMessage(message: String) {
        store(message)
      }
    })
  }

  private def webSocketUrl(): String = {
    val response = Http(slackUrl).param("token", token).asString.body
    JSON.parseFull(response).get.asInstanceOf[Map[String, Any]].get("url").get.toString
  }
```

**Figura A.22** – Código de la clase del receptor – Ejemplo de Streaming

Es un código simple que permite conectarnos a la API de una sala de chat de Slack y, en este caso, guardar lo que la gente escribe.
Este código ha sido extraído de un blog donde también explican como encontrar el token para conectarse a la API de slack.

# ANEXO IX. EJEMPLO COMPLETO SPARK, LIBRERÍA MLLIB

El ultimo ejemplo de Spark es el de la librería de Machine Learning, el código completo es el siguiente:

```
import java.io.File

import scala.io.Source

import org.apache.log4j.Logger
import org.apache.log4j.Level

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd._
import org.apache.spark.mllib.recommendation.{ALS, Rating, MatrixFactorizationModel}

object MovieLensALS {

  def main(args: Array[String]) {

    Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    if (args.length != 2) {
      println("Usage: /path/to/spark/bin/spark-submit --driver-memory 2g --class MovieLensALS " +
        "target/scala-*/movielens-als-ssembly-*.jar movieLensHomeDir personalRatingsFile")
      sys.exit(1)
    }

    // set up environment

    val conf = new SparkConf()
      .setAppName("MovieLensALS")
      .set("spark.executor.memory", "2g")
    val sc = new SparkContext(conf)

    // load personal ratings

    val myRatings = loadRatings(args(1))
    val myRatingsRDD = sc.parallelize(myRatings, 1)

    // load ratings and movie titles

    val movieLensHomeDir = args(0)

    val ratings = sc.textFile(new File(movieLensHomeDir, "ratings.dat").toString).map { line =>
      val fields = line.split("::")
      // format: (timestamp % 10, Rating(userId, movieId, rating))
      (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
    }

    val movies = sc.textFile(new File(movieLensHomeDir, "movies.dat").toString).map { line =>
      val fields = line.split("::")
      // format: (movieId, movieName)
      (fields(0).toInt, fields(1))
    }.collect().toMap
    val numRatings = ratings.count()
    val numUsers = ratings.map(_._2.user).distinct().count()
    val numMovies = ratings.map(_._2.product).distinct().count()

    println("Got " + numRatings + " ratings from "
      + numUsers + " users on " + numMovies + " movies.")

    // split ratings into train (60%), validation (20%), and test (20%) based on the
    // last digit of the timestamp, add myRatings to train, and cache them

    val numPartitions = 4
    val training = ratings.filter(x => x._1 < 6)
      .values
      .union(myRatingsRDD)
      .repartition(numPartitions)
      .cache()
    val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
      .values
      .repartition(numPartitions)
      .cache()
    val test = ratings.filter(x => x._1 >= 8).values.cache()

    val numTraining = training.count()
    val numValidation = validation.count()
    val numTest = test.count()
```

```
    println("Training: " + numTraining + ", validation: " + numValidation + ", test: " + numTest)

    // train models and evaluate them on the validation set

    val ranks = List(8, 12)
    val lambdas = List(0.1, 10.0)
    val numIters = List(10, 20)
    var bestModel: Option[MatrixFactorizationModel] = None
    var bestValidationRmse = Double.MaxValue
    var bestRank = 0
    var bestLambda = -1.0
    var bestNumIter = -1
    for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
      val model = ALS.train(training, rank, numIter, lambda)
      val validationRmse = computeRmse(model, validation, numValidation)
      println("RMSE (validation) = " + validationRmse + " for the model trained with rank = "
        + rank + ", lambda = " + lambda + ", and numIter = " + numIter + ".")
      if (validationRmse < bestValidationRmse) {
        bestModel = Some(model)
        bestValidationRmse = validationRmse
        bestRank = rank
        bestLambda = lambda
        bestNumIter = numIter
      }
    }

    // evaluate the best model on the test set

    val testRmse = computeRmse(bestModel.get, test, numTest)

    println("The best model was trained with rank = " + bestRank + " and lambda = " + bestLambda
      + ", and numIter = " + bestNumIter + ", and its RMSE on the test set is " + testRmse + ".")

    // create a naive baseline and compare it with the best model

    val meanRating = training.union(validation).map(_.rating).mean
    val baselineRmse =
      math.sqrt(test.map(x => (meanRating - x.rating) * (meanRating - x.rating)).mean)
    val improvement = (baselineRmse - testRmse) / baselineRmse * 100
    println("The best model improves the baseline by " + "%1.2f".format(improvement) + "%.")

    // make personalized recommendations

    val myRatedMovieIds = myRatings.map(_.product).toSet
    val candidates = sc.parallelize(movies.keys.filter(!myRatedMovieIds.contains(_)).toSeq)
    val recommendations = bestModel.get
      .predict(candidates.map((0, _)))
      .collect()
      .sortBy(- _.rating)
      .take(50)

    var i = 1
    println("Movies recommended for you:")
    recommendations.foreach { r =>
      println("%2d".format(i) + ": " + movies(r.product))
      i += 1
    }

    // clean up
    sc.stop()
  }

  /** Compute RMSE (Root Mean Squared Error). */
  def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n: Long): Double = {
    val predictions: RDD[Rating] = model.predict(data.map(x => (x.user, x.product)))
    val predictionsAndRatings = predictions.map(x => ((x.user, x.product), x.rating))
      .join(data.map(x => ((x.user, x.product), x.rating)))
      .values
    math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)).reduce(_ + _) / n)
  }

  /** Load ratings from file. */
  def loadRatings(path: String): Seq[Rating] = {
    val lines = Source.fromFile(path).getLines()
    val ratings = lines.map { line =>
      val fields = line.split("::")
      Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble)
    }.filter(_.rating > 0.0)
    if (ratings.isEmpty) {
      sys.error("No ratings provided.")
    } else {
      ratings.toSeq
    }
  }
```

**Figura A.23** – Código completo del ejemplo de MLlib

Recomendamos también echarle un vistazo a la fuente del código donde se indica como crear nuestro archivo de “ratings” a partir de un script de Python, que ellos mismos proporcionan, es un ejemplo muy completo para iniciarse en la librería de Machine Learning.

# ANEXO X. USO DE UNA LIBRERÍA EXTERNA PARA APACHE SPARK

Durante la última parte del trabajo trabajamos con una librería externa para Spark, la librería K-NN. El primer paso es instalar la librería en el ordenador local (o en todas las máquinas del clúster si van a usar la librería) para eso hay que conocer su repositorio que generalmente facilitará el autor de la librería. Esta parte es común para todo el mundo y sería suficiente en caso de trabajar desde línea de comandos, en nuestro caso usamos IntelIJ IDEA como IDE para escribir código, así que tenemos que encontrar la manera de indicarle que usaremos esta librería que no está incluida en Spark ni tampoco en las configuraciones SBT que se nutren de dichas librerías.

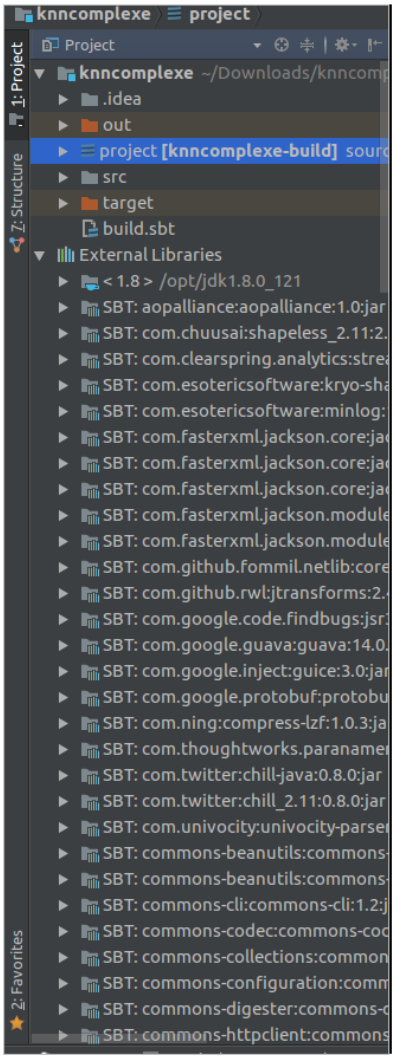



**Figura A.24** – Pestaña proyecto de IntelIJ IDEA

En la pestaña proyecto podemos ver todas las carpetas y archivos que contiene nuestro proyecto, dado que usamos SBT podemos ver todas las librerías de Spark disponibles (de las cuales solo usaremos aquellas indicadas en el archivo build.sbt).

Desde el botón archivo>estructura del proyecto, que se puede ver en la Figura A.25, seleccionaremos la opción de librerías y allí podremos añadir el .jar que contenga la librería.

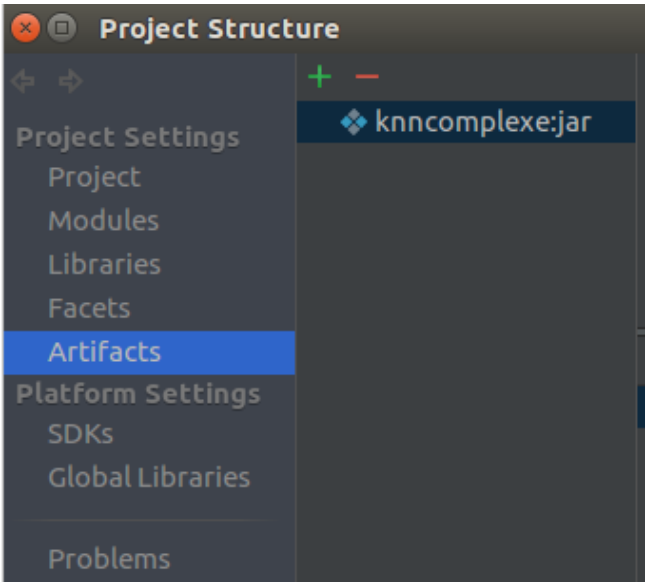



**Figura A.25** – Ventana “Project Structure” IntelIJ IDEA

En la Figura A.25 vemos otra de las cosas que cambian, hasta ahora para crear el archivo .jar lo hacíamos desde consola mediante SBT, ahora que tenemos librerías externas este método queda inutilizado así que tendremos que crear el .jar con la clase que nos interese como ‘artifact’ desde IDEA mismo, ya que no se podrá realizar con el proceso SBT común.

# ANEXO XI. EJEMPLO EN R - CÓDIGO VUELOS CON RETRASO

En esta parte del Anexo, adjuntaremos el código íntegro del primer ejemplo con R, en el que tratamos de evaluar los diferentes vuelos con retraso.

```
#Instalamos el paquete "sparklyr"
install.packages("sparklyr")
#Utilizamos el paquete previamente instalado (sparklyr)
library(sparklyr)
#Instalamos, en nuestro caso, la versión en local 2.0.0 de Spark
spark_install(version = "2.0.0.")
#Iniciamos y hacemos la conexión remota con Spark
sc <- spark_connect(master = "local")

#Utilizamos el paquete dplyr (previamente instalado)
library(dplyr)
#Copiamos los datos de R a Spark usando la función copy_to del paquete dplyr.
iris_tbl <- copy_to(sc, iris)
flights_tbl <- copy_to(sc, nycflights13::flights, "flights")
batting_tbl <- copy_to(sc, Lahman::Batting, "batting")

#Filtramos los vuelos para los retrasos
flights_tbl %>% filter(dep_delay == 2)
delay <- flights_tbl %>%
  group_by(tailnum) %>%
  summarise(count = n(), dist = mean(distance), delay = mean(arr_delay)) %>%
  filter(count > 20, dist < 2000, !is.na(delay)) %>%
  collect()

#Representamos las gráficas de los restrasos de los vuelos con el paquete ggplot2
library(ggplot2)
ggplot(delay, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth() +
  scale_size_area(max_size = 2)
```

**Figura A.26** – Código íntegro del ejemplo en R – Vuelos con Retraso

# ANEXO XII. INSTALACIÓN DETALLADA EN MODO LOCAL DE APACHE SPARK EN UBUNTU 16.04 LTS

El primer paso a realizar será asegurarnos que tenemos la última versón de Java instalada, sino es el caso, procederemos a instalarla de la siguiente manera mediante comandos por consola:

```
marc@ubuntu:~$ sudo apt-add-repository ppa:webupd8team/java
 Oracle Java (JDK) Installer (automatically downloads and installs Oracle JDK7 /
 JDK8 / JDK9). There are no actual Java files in this PPA.
```

```
marc@ubuntu:~$ sudo apt-get update
```

```
marc@ubuntu:~$ sudo apt-get install oracle-java8-installer
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gsfonts-x11 oracle-java8-set-default
Suggested packages:
```

**Figura A.27** – Añadir, actualizar e instalar el repositorio de Java en Ubuntu

Una vez realizados estos 3 pasos, tendremos que instalar la última versión de Scala que sea totalmente compatible con nuestra versión elegida de Apache Spark. En nuestro caso, hemos elegido la versión 2.1.0 de Apache Spark, que se corresponde con la versión 2.11.6 de Scala, este paso es muy importante, ya que, si instalamos otra versión puede derivar en errores futuros.

Para la realización de Scala, hemos utilizado un programa llamado Synaptic, el cuál, realiza las mismas funciones que los comandos “apt-get” pero de manera gráfica y sencilla, y hemos instalado la versión de Scala adecuada.

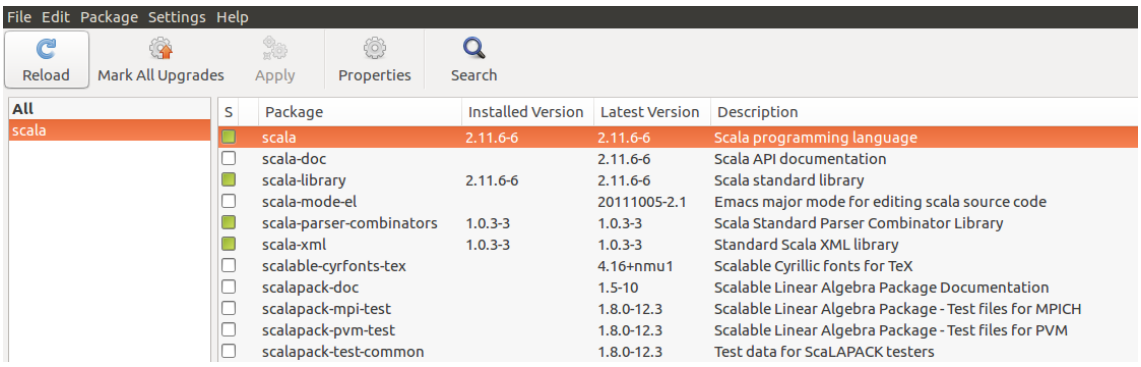



**Figura A.28** – Instalación de la versión 2.11.6 de Scala mediante Synaptic

El siguiente paso, es instalar la versión 2.1.0 de Apache Spark, para ello iremos a la página oficial de Spark y nos descargaremos dicha versión (http://spark.apache.org/downloads.html).

Pondremos el archivo .tgz descargado en la carpeta que queramos instalarlo utilizando el siguiente comando:

***sudo mv spark-2.1.0-bin-hadoop2.7.tgz /opt/spark***

Iremos al directorio indicado y los descomprimiremos allí mismo:

***cd /opt/spark***

***sudo tar -xvf spark-2.1.0-bin-hadoop2.7.tgz***

Una vez tengamos realizados todos estos pasos, un paso totalmente fundamental será indicarle al propio sistema operativo donde se encuentra la ruta del directorio de Spark para que no haya ningún tipo de problema, para ello lo añadiremos en el fichero .bashrc tal y como muestra la siguiente Figura A.29.

```
marc@ubuntu:/opt/spark/spark-2.1.0-bin-hadoop2.7/bin$ sudo nano ~/.bashrc
```

```
# Set SPARK_HOME
export SPARK_HOME=/opt/spark/spark-2.1.0-bin-hadoop2.7
export PATH=$PATH:$SPARK_HOME/bin
```

**Figura A.29** – Abrir el fichero .bashrc y añadir la ruta para encontrar la carpeta de Spark

Finalmente, probaremos que se ejecute correctamente Spark, para ello nos moveremos a la carpeta bin del propio Spark, y ejecutaremos el spark-shell para ejecutarlo por línea de comandos. Los siguientes comandos decriben lo que acabamos de explicar:

***cd /opt/spark/spark-2.1.0-bin-hadoop2.7/bin***

***sudo ./bin/spark-shell*** (este comando sirve para lanzar la consola de Spark con Scala, si queremos lanzar la consola de Spark con Python por ejemplo sería: ***sudo ./bin/pyspark***)

Una vez ejecutados, nos aparecerá la consola de Spark, tal y como muestra la Figura A.30.

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

**Figura A.30** – Ejecución exitosa de la consola de Spark

# ANEXO XIII. CÓDIGO FINAL USADO EN LA PARTE EXPERIMENTAL

En este parte del Anexo introduciremos el código usado para sacar las conclusiones finales en el apartado experimental del Capítulo 4. Dado que el código es largo y repetitivo introduciremos solo algunas de las partes, tanto de Bayes como de K-NN.

```
// Split data
val Array(training, test, validation) = data1.randomSplit(Array(0.6, 0.2, 0.2))
val numValidation =validation.count()


val lambdas = List(0, 0.1, 0.3, 0.5, 0.7, 1.0) //Trobar el millor model (millor lambda)
var milloracc = Double.MaxValue
var bestLambda = -1.0
for (lambda <- lambdas) {
  val model = NaiveBayes.train(training, lambda = lambda, modelType = "multinomial")
  val predictionAndLabel = validation.map(p => (model.predict(p.features), p.label))
  val validationacc = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / numValidation
  println("Precisió (validation) = " + validationacc + " pel model amb lambda = " + lambda)
  if (validationacc < milloracc) {
    milloracc = validationacc
    bestLambda = lambda

  }
}
```

**Figura A.31** – Data Split y selección de mejor lambda

En la Figura A.31 podemos ver como se selecciona la mejor Alpha del modelo de Bayes, un método similar se puede usar para evaluar diferentes N en el modelo de K-NN.

```
var bdng = 0
var num1= predict.filter(p=> p==0).count()
var num2= predict.filter(p=> p==1).count()
var num3= predict.filter(p=> p==2).count()
if (num1 > num2 && num1 > num3){...} //EDIFICI 0
else if (num2 > num1 && num2 > num3){...} //EDIFICI 1
else {...} //EDIFICI 2

val durada = (System.nanoTime() - t1) / 1e9d
println ("Temps=" + durada + "s")
```

**Figura A.32** – Selección de edificios

El apartado del código de la Figura A.32 muestra como se hace la primera selección para el modelo de Bayes. Además, vemos el último comando que nos muestra el tiempo total que ha necesitado el código para ejecutarse.

```
if (num1 > num2 && num1 > num3){
  bdng = 0
  println ("L'edifici es el " + bdng)
  /** PREDICCIO DEL PIS */
  val B0 = MLUtils.loadLibSVMFile(sc, "/home/jpf/Downloads/DataSets/BB0")
  var Array(training2, test2) = B0.randomSplit(Array(0.7, 0.3))
  var model2 = NaiveBayes.train(training2, lambda = 1.0, modelType = "multinomial")
  var predict2 = test2.map(p => model2.predict(p.features))
  var predictionAndLabel2 = test2.map(p => (model2.predict(p.features), p.label))
  var accuracy2 = 1.0 * predictionAndLabel2.filter(x => x._1 == x._2).count() / test2.count()
  println("accuracy dins de l'edifici 0: " + accuracy2)

  var pis = 0
  var num10= predict2.filter(p=> p==0).count()
  var num11= predict2.filter(p=> p==1).count()
  var num12= predict2.filter(p=> p==2).count()
  var num13= predict2.filter(p=> p==3).count()
  if (num10 > num11 && num10 > num12 && num10 > num13){
    pis = 0
    println("El pis es el" + pis)
    /** PREDICCIO DE LONGITUD */
    val B00 = MLUtils.loadLibSVMFile(sc, "/home/jpf/Downloads/DataSets/B0/B00")
    var Array(training3, test3) = B00.randomSplit(Array(0.7, 0.3))
    var model3 = NaiveBayes.train(training3, lambda = 1.0, modelType = "multinomial")
    var predict3 = test3.map(p => model3.predict(p.features))
    var predictionAndLabel3 = test3.map(p => (model3.predict(p.features), p.label))
    var accuracy3 = 1.0 * predictionAndLabel3.filter(x => x._1 == x._2).count() / test3.count()
    println("accuracy dins de la planta 0: " + accuracy3)

  } //PIS 0
```

**Figura A.33** – Selección de piso en Bayes

```
var pis = 0
var num10= predict2.filter($"predicted" <= 0).count()
var num11= predict2.filter($"predicted" <= 1).count() - num10
var num12= (predict2.filter($"predicted" <= 2).count() - num11) - num10
var num13= ((predict2.filter($"predicted" <= 3).count() - num12) - num11) - num10
var num14= predict2.filter($"predicted" >= 4).count()
//println ("valors: " + num10 + " " +num11 + " " + num12 + " " + num13 + " " + num14)

if (num10 > num11 && num10 > num12 && num10 > num13 && num10 > num14){
  pis = 0
  println("El pis es el" + pis)
  /** PREDICCIO DE LONGITUD */
  val B201 = MLUtils.loadLibSVMFile(sc, "/home/jpf/Downloads/DataSets/B2/B20").toDF()
  val B20 = MLUtils.convertVectorColumnsToML(B201)
  var Array(training3, test3) = B20.randomSplit(Array(0.7, 0.3))
  val knn3 = new KNNClassifier()
    .setTopTreeSize(B20.count().toInt / 500)
    .setK(10)
    .setPredictionCol("predicted")
  val knnModel3 = knn3.fit(training3)
  val predict3 = knnModel3.transform(test3)
  val accuracy3 = validate(predict3)
  println("accuracy a la planta 0: " + accuracy3)
```

**Figura A.34** – Selección de piso en K-NN

En las Figuras A.33 y A.34 observamos casi la misma sección de código para Bayes y K-NN, las diferencias parecen mínimas, pero es crítico que sean precisas. Esta sección del código, de manera similar a como se hace anteriormente con el edificio, predice el piso donde se encuentra la persona y finalmente, dentro del piso, predice las coordenadas.

# ANEXO XIV. USO DE SPARKLING WATER (H2O) MACHINE LEARNING EN R

En esta parte veremos un poco por encima el funcionamiento del paquete "*rsparkling*" que nos proporciona enlaces a algoritmos de H2O a través de "*sparklyr*". Si utilizamos la versión más actualizada de Spark, deberemos también utilizar la última versión de "*rsparkling*".

Si hacemos una comparación con la utilización de Spark Machine Learning Library (MLlib), veremos que las funciones y algoritmos son prácticamente los mismos, con sus respectivas variaciones de código eso sí. En definitiva, podremos hacer las mismas predicciones utilizando herramientas de H2O que con las de MLlib.

Por tanto, a efectos prácticos utilizándolo con R, no notaremos grandes variaciones ya sea utilizando una u otra, las predicciones serán muy parecidas y aproximadas.

Para ello hemos elegido el mismo ejemplo que utilizamos en MLlib, el ejemplo de Regresión Lineal (adjuntado en este mismo Anexo V), en el que comparábamos el consumo real de combustible contra la predicción. Evidentemente, el resultado final (gráfico) será muy parecido al que ya habíamos obtenido, así que nos centraremos en las principales diferencias de código relevantes, pero explicaremos las partes más importantes.

La primera parte constará en utilizar las librerías adecuadas para poder utilizar H2O (Figura A.35):

```
# Instalamos el paquete rsparkling
install.packages("rsparkling")

# Fijamos la versión de rsparkling (2.0.3) ya que utilizamos la última versión de Spark (2.1.0)
options(rsparkling.sparklingwater.version = "2.0.3")

# Cargamos las librerías que nos harán falta
library(rsparkling)
library(h2o)
library(dplyr)
library(sparklyr)

# Iniciamos la conexión con Spark y copiamos las tablas
sc <- spark_connect("local")
mtcars_tbl <- copy_to(sc, mtcars, "mtcars")
```

**Figura A.35** – Utilización de las librerías para hacer uso de H2O

En la segunda parte transformaremos los datos mediante H2O para poder trabajar con ellos, dividiremos los datos dentro de H2O en lugar de Spark y ajustaremos nuestro modelo de H2O para predecir el consumo medio de combustible y compararla con el consumo real. Tal y como muestra la Figura A.36:

```
# Convertimos nuestros datos en datos H2O utilizando funciones de rsparkling
training <- as_h2o_frame(sc, partitions$training, strict_version_check = FALSE)
test <- as_h2o_frame(sc, partitions$test, strict_version_check = FALSE)

# Ajustamos un modelo lineal al conjunto de datos
glm_model <- h2o.glm(x = c("wt", "cyl"),
                     y = "mpg",
                     training_frame = training,
                     lambda_search = TRUE)
print(glm_model)

library(ggplot2)

# Calcular los valores pronosticados en nuestro conjunto de datos de prueba
pred <- h2o.predict(glm_model, newdata = test)
# convertimos de formato H2O a Spark los datos
predicted <- as_spark_dataframe(sc, pred, strict_version_check = FALSE)
```

**Figura A.36** – Funciones utilizando H2O con rsparkling

Finalmente, el gráfico resultante será prácticamente igual que el del Gráfico A.17. El uso de la librería con H2O en R no dista mucho de los resultados obtenidos con MLlib (la librería de serie que viene con Spark). Tal vez utilizando la suite por separado de H2O si que salga más a cuenta que la de serie, pero utilizándolo conjuntamente con R no vemos ninguna ventaja al respecto.

Nosotros recomendamos la utilización de MLlib, ya que no tenemos tantos problemas de librerías (H2O tiene más y se pueden dar errores), las funciones son más sencillas de utilizar y de entender, ya viene por defecto integrada con Spark y tiene la totalidad de algoritmos para hacer diferentes predicciones.

# ANEXO XV. USO DE LA LIBRERÍA SPARKR

Otra de las librerías que podemos utilizar juntamente con Spark es SparkR. Esta librería en las últimas versiones de R (en las muy antiguas como la 1.6.0 sigue apareciendo como paquete) ya no es un paquete como tal, y, por tanto, ya no se puede instalar ni usar como tal. Si queremos hacer uso de ella, tenemos que tener instalado Spark en nuestra máquina y hacer la conexión manualmente pasando la ruta donde se encuentra.

En nuestro caso al utilizar la última versión de R (3.4.0) hemos tenido que hacerlo así, y aquí os mostramos el código para establecer esta conexión manual (Figura A.37):

```
Sys.setenv(SPARK_HOME = "/opt/spark/spark-2.1.0-bin-hadoop2.7/")
.libPaths(c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib"), .libPaths()))
library(SparkR)
sparkR.session(master = "local[*]", sparkConfig = list(spark.driver.memory = "2g"))
```

**Figura A.37** – Conexión manual con Spark para poder ejecutar SparkR en R

Como se puede observar en la anterior Figura A.37, en SPARK_HOME le pasaremos toda la ruta donde tenemos instalado Spark, y una vez la encuentre, lo cargará en RStudio y podremos hacer uso de la librería, iniciar la conexión e indicarle cuanta memoria queremos que utilice.

A pesar del uso de la conexión manual en la versión más reciente de R, SparkR nos proporciona todas y cada una de las funcionalidades de Sparklyr, exactamente las mismas, por tanto, no creemos oportuno realizar ningún ejemplo ya que serían iguales o muy parecidos a los anteriores. Queremos destacar que también existe esta herramienta para conectar con Spark y para realizar todo lo anteriormente comentado, aunque creemos que es mucho más cómoda la librería Sparklyr ya que está totalmente actualizada, no tiene problemas de versiones y es más cómoda de utilizar ahora mismo.