Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

IMPLEMENTACIÓN DE MODELOS DE DATA SCIENCE UTILIZANDO SPARK Y MLLIB

Alumno: Patricio Serrano Expósito

Tutores: Prof. D. Antonio Jesús Rivera Rivas
Prof. Dña. María Dolores Pérez Godoy

Dpto: Informática

3 de septiembre, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Antonio Jesús Rivera Rivas y Dña. María Dolores Pérez Godoy, tutores del Proyecto Fin de Carrera titulado: IMPLEMENTACIÓN DE MODELOS DE DATA SCIENCE UTILIZANDO SPARK Y MLLIB, que presenta Patricio Serrano Expósito, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

RIVERA RIVAS ANTONIO JESUS - 26015863B

Firmado digitalmente por RIVERA RIVAS ANTONIO JESUS - 26015863B
Fecha: 2019.09.03 11:04:19 +02'00'

PEREZ GODOY MARIA DOLORES - 75093149N

Firmado digitalmente por PEREZ GODOY MARIA DOLORES - 75093149N
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-75093149N, givenName=MARIA DOLORES, sn=PEREZ GODOY, cn=PEREZ GODOY MARIA DOLORES - 75093149N
Fecha: 2019.09.03 11:11:01 +02'00'

Índice

1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS DEL PROYECTO

En esta sección se introduce el trabajo de fin de grado. Este es un trabajo de investigación donde se han implementado los algoritmos de ensembles de ciencia de datos *Stacking* y *Bagging* utilizando Apache Spark y su biblioteca de *machine learning* MLlib.

1.1 Introducción

En la actualidad, el mundo en que vivimos se encuentra cada vez más interconectado. Cada vez se encuentran más presentes en nuestras vidas dispositivos conectados a Internet que recogen todo tipo información, desde los más comunes como ordenadores, móviles o televisiones hasta electrodomésticos inteligentes, relojes o asistentes personales controlados por voz.

La gran cantidad de datos que generan estos dispositivos puede ser analizada y utilizada para extraer conocimiento a partir de ellos, pero el volumen de estos datos, la velocidad a la que se generan y su variedad diariamente, hacen difícil el uso de herramientas y técnicas tradicionales para la extracción de conocimiento de ellos. Con el objetivo de hacer uso de estos datos surgen los campos de ciencia de datos (Data Science) [Edureka2019] y Big Data [IBM2015].

Figura 1.1 Qué ocurre en internet en 60 segundos

Fuente: go-globe.com

Una de las áreas de la ciencia de datos es *machine learning* [AnalyticsVidhya2018], que consiste en la construcción de modelos que permiten aprender de los datos.

Este trabajo se centra en los campos de ciencia de datos y *machine learning* utilizando grandes cantidades de datos para crear modelos que sean capaces de obtener conocimiento a partir de dichos datos. En este caso, el conocimiento extraído se utilizará para realizar predicciones de nuevos datos, entendiendo por predicción las tareas de clasificación, regresión y predicción de series temporales.

El objetivo de este trabajo es desarrollar *ensembles* [Diettrich2000], modelos que combinan las predicciones de varios modelos base, de forma que se mejore el rendimiento de los modelos iniciales.

1.2 Motivación

En el anterior apartado, se destaca la importancia de tratar con cantidades de datos tan grandes que no es posible hacerlo con herramientas tradicionales. Por este motivo surgen las herramientas Big Data que son capaces de tratar con estas cantidades de datos. Entre ellas destaca el *framework* Apache Spark [Kovachev2019], el cual se utiliza en este trabajo por su gran cantidad de funcionalidades y su eficiencia al procesar datos.

Apache Spark contiene un módulo llamado MLlib con implementaciones de múltiples modelos de *machine learning.* Este módulo es de interés para este trabajo porque los modelos que contiene están desarrollados y son parametrizables por lo que permite centrarse en la creación e implementación de los ensembles sin tener que crear los modelos desde cero.

Las principales motivaciones que me han llevado a la realización de este proyecto son:

- Interés en realizar algún proyecto relacionado con ciencia de datos, Big Data y *machine learning*, especialmente por la importancia que están tomando estas disciplinas en la actualidad.
- El uso de una tecnología novedosa como Apache Spark, que permite el tratamiento de enormes cantidades información de las que se ha hablado anteriormente.
- Crear una implementación de los algoritmos de ensembles *Bagging* [Dietrich2000, Brownlee2016, Singh2018] y *Stacking* [Dzeroski&Zenko2004, Singh2018], ya que estos no se encuentran en la biblioteca de Apache Spark MLlib.
- La posibilidad de aprender Scala, un lenguaje que se centra en la programación funcional, con la que no había tenido contacto hasta ahora.

1.3 Objetivos del proyecto

El propósito de este trabajo es la implementación de algoritmos de ensembles de ciencia de datos utilizando dos de las principales técnicas como son *Stacking* y *Bagging.* Para ello se utilizará MLlib, la biblioteca de *machine learning* de Apache Spark, utilizando Scala como lenguaje de programación.

Los objetivos a realizar en este proyecto son los siguientes:

- Estudiar el estado actual de las herramientas de Big Data y las técnicas de ciencia de datos.
- Conocer las herramientas disponibles para abordar la escalabilidad en Big Data, en particular para el entorno de programación Apache Spark.
- Instalación y configuración de los sistemas necesarios para implementar diferentes modelos de ciencia de datos y su posterior ejecución.
- Familiarización con la biblioteca actual de *machine learning* MLlib.
- Implementación de algoritmos de ciencia de datos en Spark.
- Analizar las mejoras obtenidas al implementar los algoritmos de ensembles comparados con la ejecución de modelos en solitario.

1.4 Metodología desarrollada

La metodología seguida en el desarrollo del trabajo ha sido la siguiente:

- Revisar documentación técnica actualizada sobre aspectos relacionados con la ciencia de datos.
- Instalación y configuración de las herramientas y bibliotecas a utilizar y comparación con otros sistemas existentes.
- Implementación de los algoritmos seleccionados siguiendo las directrices de los marcos de programación seleccionados.
- Evaluación de las prestaciones sobre conjuntos de datos en Big Data.
- Redacción de la memoria descriptiva del proceso de diseño, implementación y evaluación de los algoritmos.

1.5 Estructura de la memoria

Esta documentación está distribuida en múltiples secciones, de las cuales se va a realizar una introducción.

- En esta primera sección se da una visión general del proyecto con una introducción, la motivación del mismo y su planificación.
- En la segunda sección se presenta el estado actual de la ciencia de datos, Big Data y la minería de datos.
- En la tercera sección se describe el *framework* Apache Spark para el procesamiento de datos y de sus herramientas y funcionalidades.
- En la cuarta sección se introducen de forma teórica e ilustrada los dos algoritmos de ensembles que se tratan es este proyecto: Stacking y Bagging.
- En la quinta sección se explican las decisiones de diseño e implementación que se han tomado para este proyecto.
- En la sexta sección se muestran y se hace un análisis de los resultados obtenidos de los ensembles, comparándolos con la ejecución de los modelos individuales.
- En la séptima sección se muestran las conclusiones obtenidas tras la finalización de este proyecto.

Al final del documento se incluye un anexo describiendo las herramientas y el entorno de desarrollo que se han utilizado para la realización de este proyecto.

1.6 Planificación del proyecto

Éste proyecto se ha desarrollado durante un periodo de 6 meses, desde finales de enero hasta finales de agosto. Se ha elaborado un diagrama de Gantt como se puede ver en la Figura 1.2 donde se muestran las actividades realizadas a lo largo del desarrollo del proyecto y la duración que han tenido.

El desarrollo de ambos algoritmos de ensembles sobre los que trata este proyecto se ha podido realizar simultáneamente ya que estos tienen una base común.

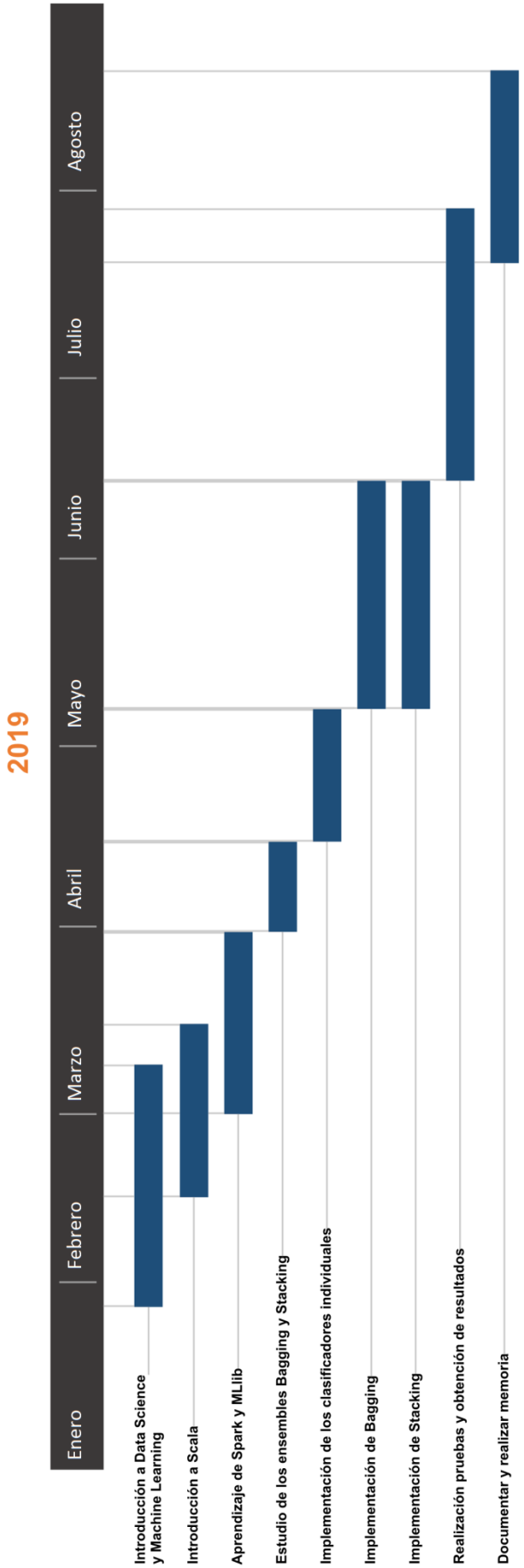

Figura 1.2 Diagrama de Gantt

2. BIG DATA Y CIENCIA DE DATOS

En esta sección se describen los campos Big Data y ciencia de datos y su situación en la actualidad. En la figura 2.1 se muestra un esquema de cómo se relacionan entre sí y con otros campos.

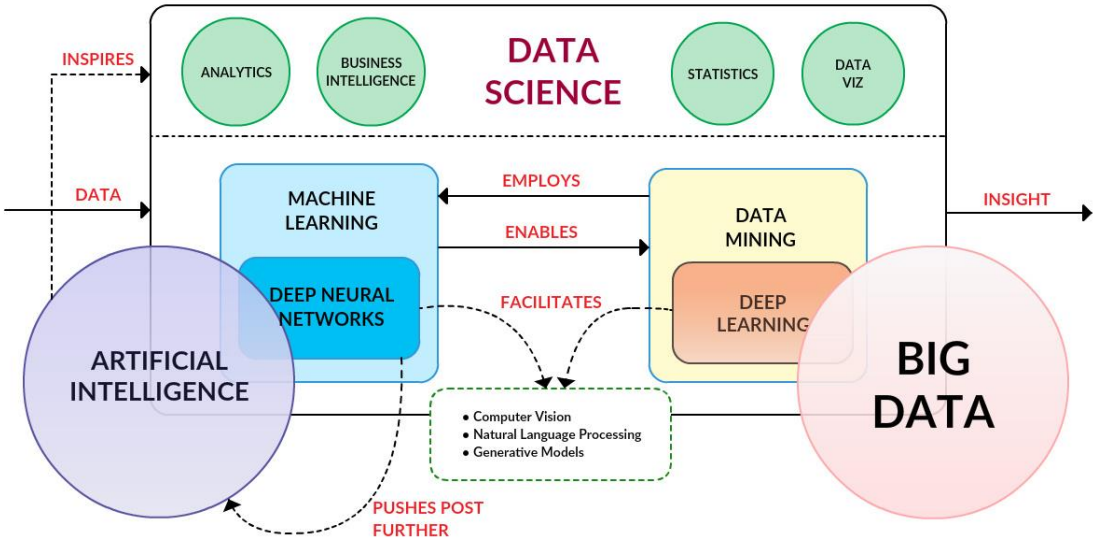

Figura 2.1 Relación entre las múltiples disciplinas

Fuente: houseofbots.com

2.1 Big Data

Cuando se habla de Big Data nos referimos a grandes conjuntos de datos o combinaciones de conjuntos de datos cuyo tamaño, diversidad y velocidad de crecimiento dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales, tales como bases de datos relacionales y estadísticas clásicas, en un tiempo razonable para que sean útiles.

Una de las causas de La complejidad del Big Data se debe a la naturaleza no estructurada de gran parte de los datos generados por las tecnologías modernas, como son los sensores incorporados en dispositivos, televisores, vehículos, ordenadores, videoconsolas, las redes sociales, smartphones, dispositivos GPS, relojes inteligentes etc.

El Big Data viene definido por diferentes V, como son:

- Volumen: es la característica más representativa del Big Data. Datos en algunos formatos como video, música e imágenes en cantidades enormes son capaces de ocupar terabytes y petabytes en bases de datos y sistemas de almacenamiento.
- Variedad: una de las características principales que determinan la calidad de los datos recopilados es su variedad. El Big Data no posee un formato único ni proviene de una sola fuente, sino que puede tratarse de imágenes, correos electrónicos, vídeos… La diversidad en los datos es muy importante para cubrir un rango amplio de casuísticas. Esta diversidad puede condicionar las metodologías con las que se analizan los datos.
- Velocidad: ya que el Big Data es la capacidad de grandes conjuntos de datos analizar datos en un tiempo aceptable, el proceso en su conjunto debe ser ágil y rápidamente transitable en cuanto a las diferentes fases que van desde la recopilación de datos al retorno del análisis que se deriva de estos.
- Veracidad: la veracidad de los datos se refiere a la precisión y si es fiable un conjunto de datos. Cuando se trata esto grandes conjuntos datos, no se trata solo de la calidad en sí, sino de cuán confiable es la fuente, el tipo y el procesamiento de los datos. Eliminar cosas como sesgos, anomalías o inconsistencias, duplicación y volatilidad son solo algunos aspectos que influyen en la mejora de la precisión de los datos.

Aunque a esta definición también se le añaden en ocasiones otras V como variabilidad, valor, vaguedad o validez.

En la figura 2.2 se puede ver con qué tratan 3 de las V anteriores: el volumen, la variedad y la velocidad.

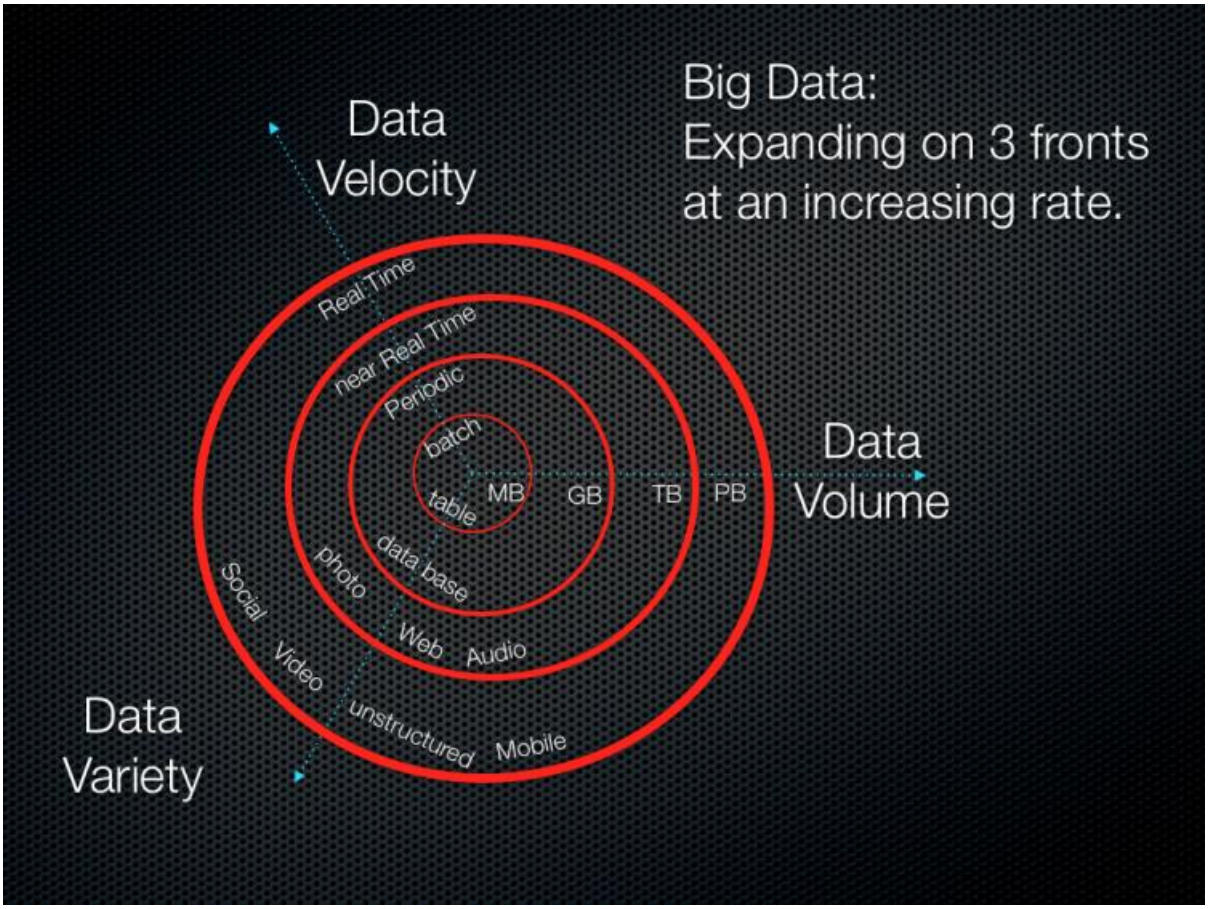

Figura 2.2 3 de las V de Big Data

Fuente: datasciencecentral.com

Lo que hace que el Big Data sea tan útil en la actualidad es el hecho de que proporciona respuestas a muchas preguntas que las empresas ni siquiera sabían que tenían. Con una cantidad tan grande de información, los datos pueden ser moldeados o probados de cualquier manera que la empresa considere adecuada. Al hacerlo, las organizaciones son capaces de identificar los problemas de una forma más comprensible.

2.2 Ciencia de datos y minería de datos

La ciencia de los datos o data science es un conjunto de herramientas que permiten extraer conocimiento a partir de los datos. Es un campo interdisciplinar que engloba competencias de estadística, matemáticas, programación, minería de datos [Educba2019] y visualización de datos.

El principal objetivo de la ciencia de datos es la obtención de un conocimiento que sea útil para los expertos del área del problema en cuestión.

La ciencia de datos consta de varios procesos (Figura 2.3) que son:

- Recopilación y extracción de datos.
- Limpieza y reestructuración de los datos para que sean aptos para ser analizados.
- Minería de datos: preprocesamiento, análisis exploratorio, creación y optimización de modelos, análisis predictivos y estadística.
- Comunicar las conclusiones obtenidas de los datos de manera eficiente al público al que están dirigidas.

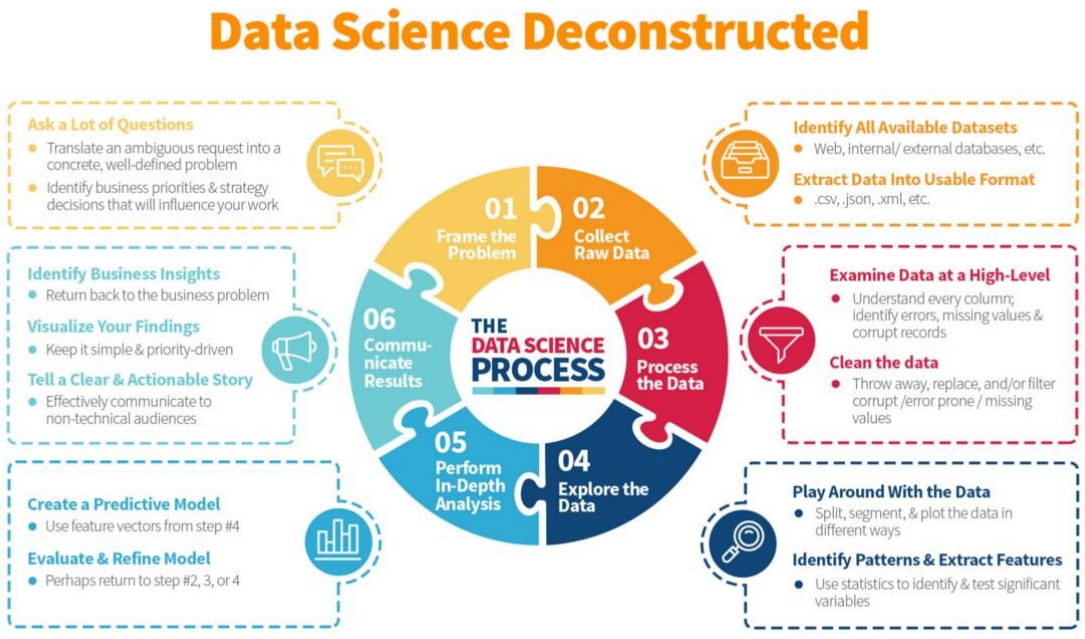

Figura 2.3 Procesos de ciencia de datos

Fuente: businessinsider.com

En el proceso de minería de datos se busca la extracción de patrones y conocimiento de grandes cantidades de datos. Para ello existen múltiples tipos de tareas.

Las tareas de minería de datos predictivas crean un modelo del conjunto de datos disponible para predecir valores desconocidos o futuros. Las tareas predictivas se dividen en:

- Clasificación: la clasificación es el proceso de encontrar un modelo que describe las clases de datos. El propósito es poder usar este modelo para predecir la clase de objetos a partir delos datos suministrados.
- Regresión: a diferencia de la clasificación, la regresión no predice una clase, sino un valor numérico. Para ello, se busca una función que establezca una correspondencia entre los datos y el valor a predecir.
- Predicción de series temporales: busca analizar datos de series de tiempo con el fin de extraer patrones, tendencias, reglas y estadísticas útiles.

Las tareas de minería de datos descriptivas generalmente encuentran datos que describen patrones y presentan información nueva y significativa del conjunto de datos disponible. Las tareas descriptivas se dividen en:

- Agrupamiento: se conoce también como clustering. Trata de clasificar elementos en distintos grupos siguiendo algún criterio.
- Descubrimiento de reglas de asociación: busca descubrir asociaciones y relaciones entre los datos.
- Sumarización de los datos: es el proceso de encontrar subconjuntos de datos que contengan la información de todo el conjunto.

En este trabajo se hace uso de las siguientes técnicas para clasificación: *Regresión Logística*, *Decision Tree* y *Naive Bayes*.

Algunas aplicaciones de la ciencia de datos son:

- Mejorar los resultados que proporcionan los motores de búsqueda como Google.
- Crear sistemas de recomendación para que a los usuarios se les muestre contenidos relevante a sus intereses en plataformas como Youtube, Amazon o Netflix.

- Detección de elementos en imágenes, como personas que se encuentran en una foto.
- Reconocimiento de voz.
- Detección de fraudes y riesgos económicos.

2.3 Técnicas de minería de datos

En este apartado se presentan las técnicas de minería de datos de clasificación que se utilizan en este trabajo.

Decision Tree

Decision Tree, conocido como árbol de decisión, es uno de los modelos más intuitivos que existen en la minería de datos ya que son fáciles de interpretar y usar.

Decision Tree crea modelos de clasificación o regresión en forma de estructura de árbol. Desglosa un conjunto de datos en subconjuntos cada vez más pequeños, mientras que al mismo tiempo se desarrolla un árbol de decisión asociado de forma incremental. El resultado final es un árbol con nodos de decisión y nodos hoja. Los nodos hoja representan una clase o decisión. El nodo de decisión superior en un árbol que corresponde al mejor predictor llamado nodo raíz.

Regresión Logística

La regresión logística es un algoritmo de clasificación utilizado para asignar observaciones a un conjunto discreto de clases. Predice la probabilidad de ocurrencia de una clase ajustando los datos a partir de una función logit.

Naive Bayes

Naive Bayes es un algoritmo de clasificación para problemas de clasificación binarios y de múltiples clases. Se basa en el teorema de Bayes suponiendo la independencia de los datos entre sí.

3. APACHE SPARK

Apache Spark es un *framework* open-source de herramientas Big Data y computación en clúster de propósito general preparado para trabajar con grandes volúmenes de datos con tolerancia a fallos.

Figura 3.1 Logo de Apache Spark

3.1 Introducción

Spark es un proyecto que empezó a ser desarrollado en 2009 en la Universidad de California Berkeley por el grupo de investigación AMPLab, centrado en la minería de datos y Big Data. El objetivo del proyecto era proporcionar alterativas a MapReduce debido a la ineficiencia de este para algunas aplicaciones de ciencia de datos, como algoritmos iterativos o algoritmos que requieren que se comparta información entre sus operaciones.

En 2010 se convirtió en un proyecto open-source bajo una licencia BSD y en 2013 el proyecto fue donado a la *Apache Software Foundation*, convirtiéndose en 2014 en uno de sus proyectos “Top-level” [Xin2014].

Desde entonces el proyecto ha estado en continuo desarrollo, alcanzando 1000 contribuidores en 2015 y más de 1900 a principios de 2019 [Phatak2015]. En la figura 3.2 se puede ver la actividad de los contribuidores en GitHub.

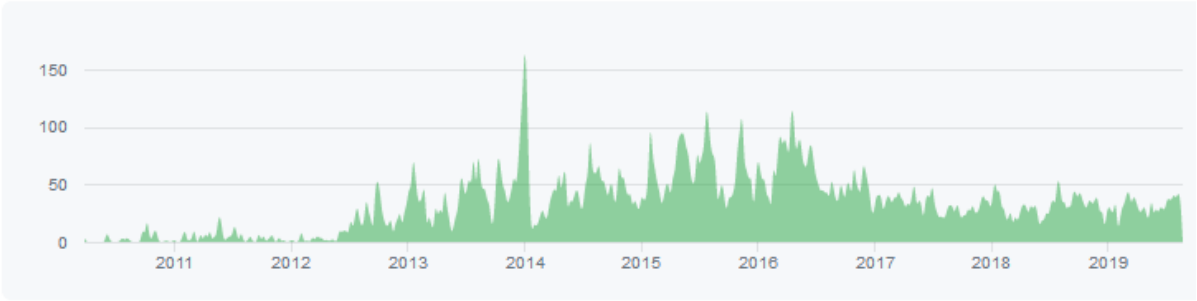

Figura 3.2 Historial de commits de Apache Spark en GitHub

Fuente: github.com

Actualmente Apache Spark lo utilizan una gran variedad de empresas y organizaciones, como Amazon, Microsoft, Facebook, Cisco, IBM, Yahoo y Oracle. En la Figura 3.3 se puede ver como Apache Spark es el proyecto más activo del lenguaje de programación Scala en GitHub.

También destaca el que algunas empresas crean sus propias herramientas para añadir funcionalidades a Apache Spark para trabajar con sus productos, como es el caso de *Microsoft Machine Learning for Apache Spark,* que es el tercer proyecto que aparece en la Figura 3.3.

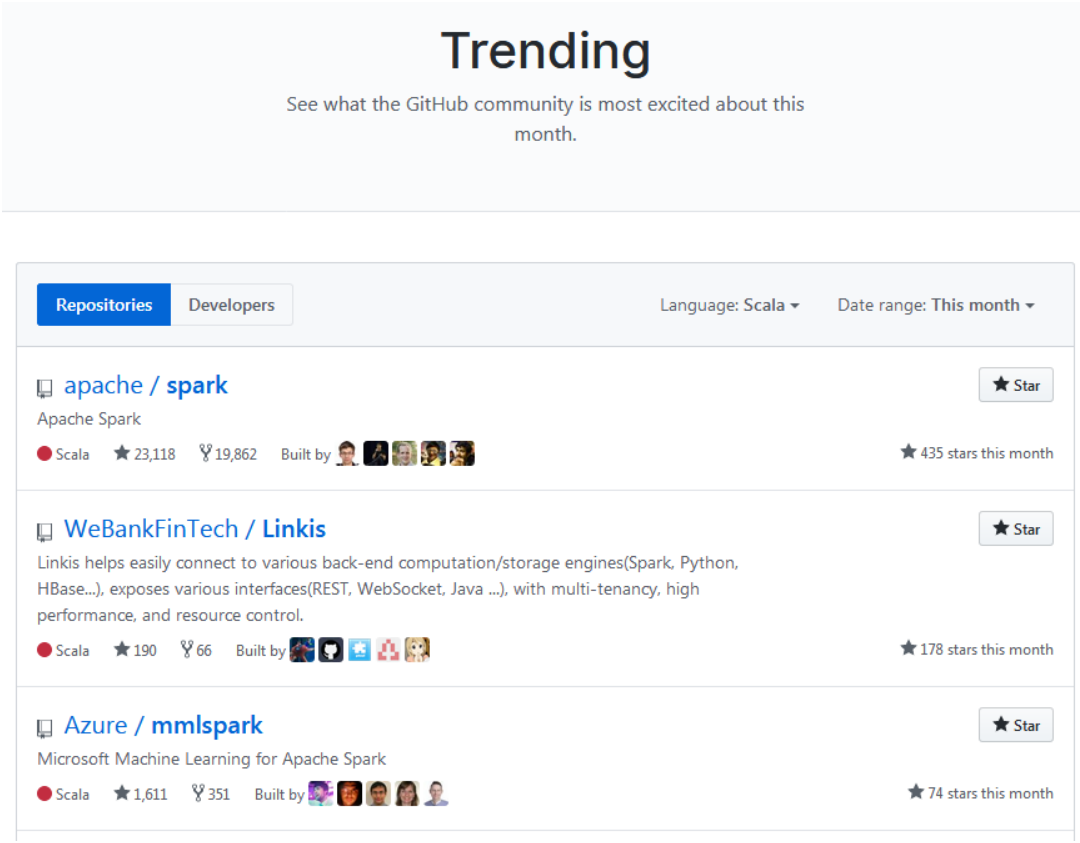

Figura 3.3 Ranking de proyectos en Scala de GitHub en Agosto de 2019

Fuente: github.com

3.2 Funcionalidades de Apache Spark

Apache Spark se presenta a sí mismo con las siguientes funcionalidades:

- Velocidad: Apache Spark es capaz de ejecutar aplicaciones hasta 100 veces más rápido si están cargadas en memoria y hasta 10 veces más rápido si están cargadas en disco que Hadoop ya que reduce el número de operaciones de lectura y escritura necesarias y haciendo más uso de la memoria para guardar datos que se van a utilizar después.
- Facilidad de uso: Apache Spark soporta los lenguajes de programación Java, Python, Scala y R, pudiéndose realizar aplicaciones en cualquiera de ellos según las preferencias de los desarrolladores. Además proporciona operadores de alto nivel en sus clases RDD y Dataframe y ofrece una Shell interactiva para Scala, Python y R.
- Generalidad: Apache Spark incluye varias bibliotecas con múltiples funcionalidades y estas pueden combinarse entre ellas sin dar ningún problema. Estas bibliotecas se describen en el apartado 3.3.
- Ejecución en cualquier sitio: Apache Spark puede ser ejecutado en Hadoop YARN, Apache Mesos, Kubernetes, en modo clúster único o en la nube. Además, permite acceder a datos de distintas fuentes como bases de datos tradicionales como MySQL y PostgreSQL, bases de datos NoSQL como Apache Cassandra y MongoDB, fuentes de datos de Hadoop como HDFS, plataformas de streaming y muchas más fuentes.

Apache Spark también ofrece tolerancia a fallos ya que su clase RDD replica la información en los varios nodos del clúster y guarda qué operaciones ha ido realizando durante las ejecuciones.

3.3 Estructura de Apache Spark

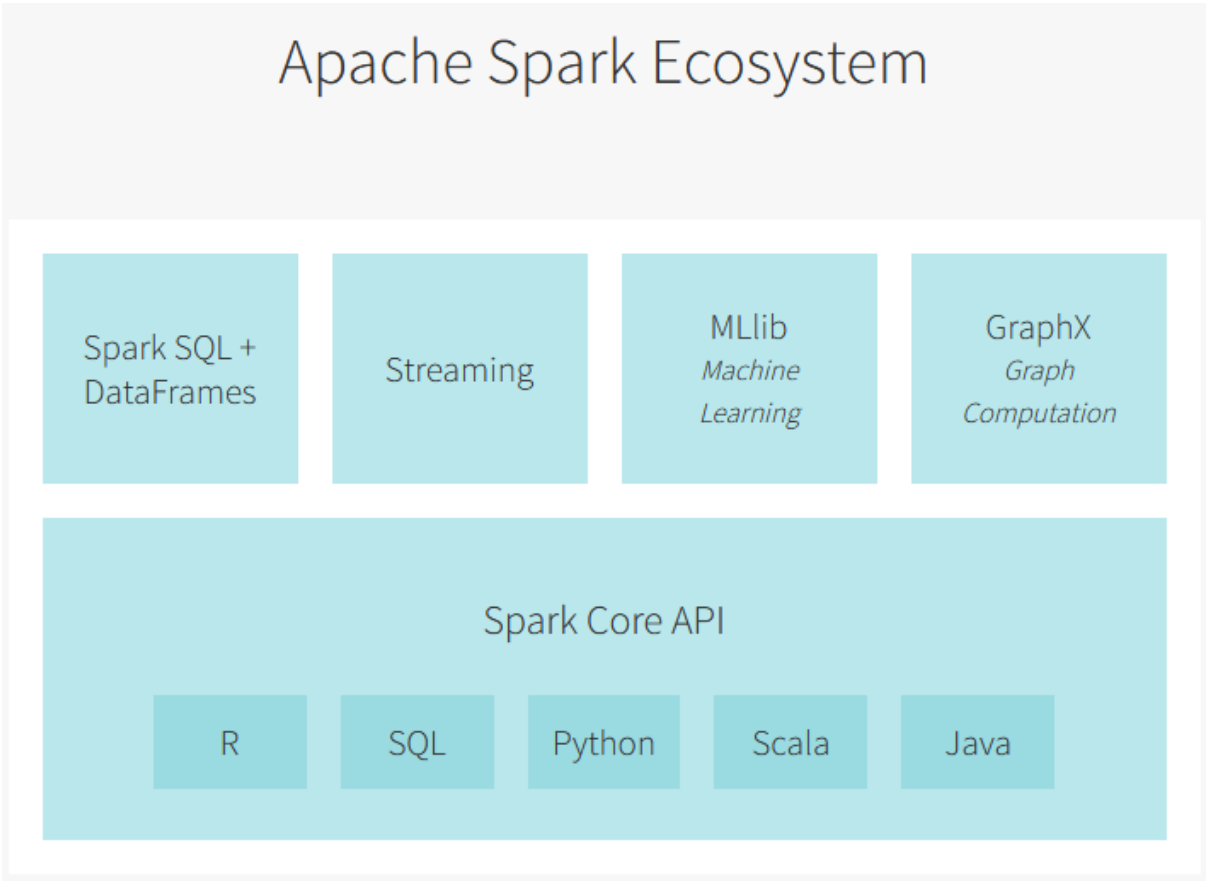

Figura 3.4 Componentes del ecosistema de Apache Spark

Fuente: databricks.com

Apache Spark está formado por los siguientes componentes que se pueden ver en la Figura 3.4:

- Spark Core: es el motor de ejecución sobre el cual están construidos los demás componentes. Tiene funcionalidades para realizar computación en memoria y garantizar la velocidad, un modelo generalizado de ejecución para dar soporte a múltiples aplicaciones y una API para los lenguajes Java, Python, Scala y R; centrada en la clase RDD y la abstracción que permite.
- Spark SQL y Dataframes: es el componente que permite realizar procesamiento de datos estructurados. Provee de la clase Dataframe, permite utilizar SQL a través de varis interfaces de línea de comandos y puede actuar como un motor de consultas SQL distribuido. Tiene múltiples funcionalidades para integrarse con el resto de componentes.

- Spark Streaming: este componente hace uso de la capacidad de scheduling de Spark Core para tratar con lotes de datos de tamaño reducido permitiendo hacer análisis de datos a tiempo real. Viene preparado para integrarse con fuentes de datos populares como Twitter.

- MLlib: es la biblioteca de *machine learning* de Apache Spark. Incluye una gran cantidad de algoritmos estadísticos y de *machine learning* en los lenguajes Java, Python y Scala implementados para trabajar tanto con RDD como con Dataframe. Ofrece también herramientas y utilidades para hacer más fácil la creación de procesos de *machine learning* (selección de datos, transformación, evaluación de los modelos etc.) [Meng2015].

- GraphX: es un motor de computación para grafos utilizando la clase Graph, basada en la clase RDD. Provee de operadores optimizados para la computación con grafos, una gran colección de algoritmos y múltiples formas de construir grafos para simplificar las tareas de análisis de grafos.

3.4 Clase RDD

La clase RDD es una de las dos principales clases de Apache Spark que sirve como abstracción de los datos. RDD significa *Resilient Distributed Dataset*, y es una colección inmutable de elementos (*Dataset*), particionado a lo largo del clúster para permitir computación en paralelo (*Distributed*), y que es tolerante a fallos. (*Resilient*)

Todos los datos que contiene un RDD son almacenados en memoria principal y las operaciones que se realizan sobre los mismos pueden realizarse también en memoria (una de la características de Apache Spark que lo hace tan rápido). Si el tamaño de los datos es demasiado grande y no caben en memoria Apache Spark hará uso del disco.

Los RDD pueden ser creados con cualquier tipo de objeto o clase creada por el desarrollador, permitiendo mucha flexibilidad para trabajar con ellos.

Sobre un RDD pueden ser aplicadas dos tipos de operaciones, las transformaciones y las acciones. (Figura 3.5)

- Las **transformaciones** son operaciones sobre uno o varios RDD cuya salida crea un nuevo RDD.

- Las **acciones** son operaciones que se ejecutan sobre un RDD teniendo en cuenta todas sus transformaciones y devuelven un valor al programa driver que se está ejecutando o almacenan el resultado en un sistema de almacenamiento externo.

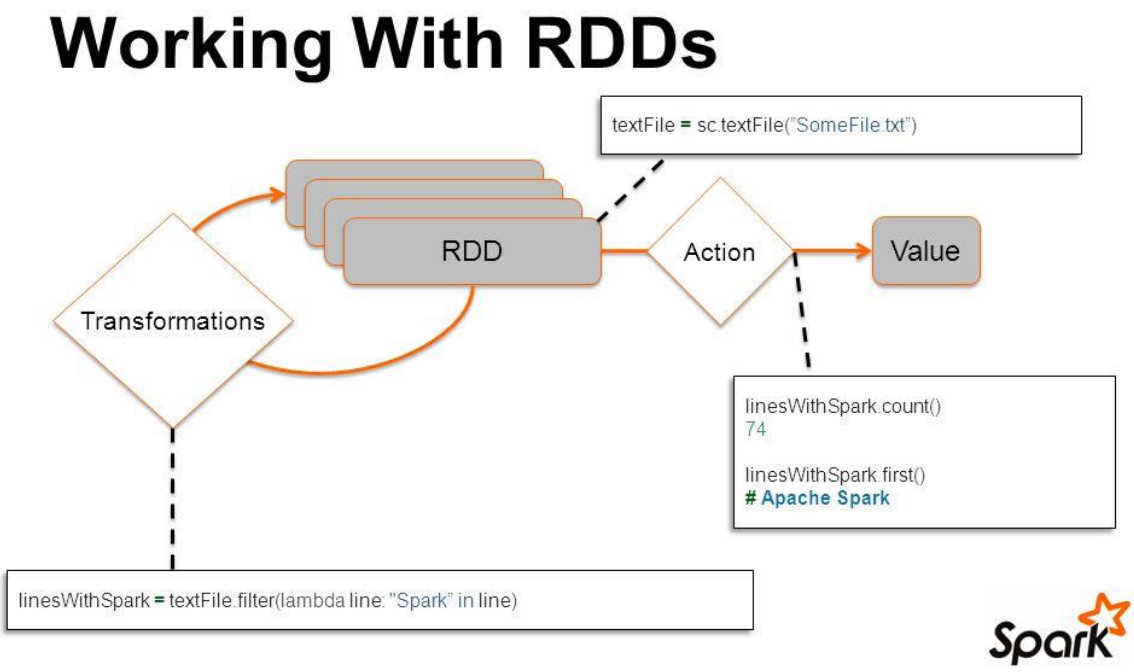

Figura 3.5 Ejemplo de operaciones: transformación y acción

Fuente: Documentación de Apache Spark

En las tablas 3.1 y 3.2 se incluyen algunos ejemplos de transformaciones y acciones comunes:

Transformación	Resultado
map(*función*)	Crea un nuevo RDD formado al aplicar *función* a todos los elementos del RDD original.
filter(*función*)	Crea un nuevo RDD formado por los elementos del RDD original que devuelven *true* al aplicar *función*.
flatMap(*función*)	Crea un nuevo RDD formado al aplicar *función* a todos los elementos del RDD original. El resultado de aplicar *función* es un array de elementos que se añaden al nuevo RDD.
union(*RDD*)	Crea un nuevo RDD que contiene la unión de los elementos del RDD original y el RDD pasado como argumento.
intersection(*RDD*)	Crea un nuevo RDD que contiene la intersección de los elementos del RDD original y el RDD pasado como argumento.

Tabla 3.1 Ejemplos de transformaciones comunes

Fuente: Elaboración propia

Acción	Resultado
count()	Devuelve el número de elementos del RDD.
collect()	Devuelve los elementos de RDD como un array.
take(*n*)	Devuelve un número *n* de elementos del RDD.
foreach(*función*)	Ejecuta *función* en cada uno de los elementos del RDD.
reduce(*función*)	Agrega los elementos del RDD mediante *función*. *función* debe ser un operador conmutativo y asociativo que acepte dos elementos y devuelva uno.

Tabla 3.2 Ejemplos de acciones comunes

Fuente: Elaboración propia

Todas las transformaciones en Apache Spark son *lazy*, ya que no calculan sus resultados de inmediato. En cambio, solo recuerdan las transformaciones aplicadas a algún conjunto de datos base (por ejemplo, un archivo). Las transformaciones solo se calculan cuando una acción requiere que se devuelva un resultado al programa driver. Este diseño permite que Apache Spark funcione de manera más eficiente. Por ejemplo, permite a Apache Spark darse cuenta de que un dataset creado a través del *map* se usará en un *reduce* y devolverá solo el resultado de *reduce* al programa, en lugar del dataset de *map* más grande.

Por defecto, cada RDD transformado tiene que ser recalculado cada vez que se ejecuta una action en él. Sin embargo, también se puede conservar un RDD en memoria utilizando los métodos *persist* o *cache*, en cuyo caso Spark mantendrá los elementos en el clúster para un acceso mucho más rápido la próxima vez que lo consulte. También existe soporte para RDD persistentes en disco.

3.5 Clase Dataframe

La clase Dataframe es otra clase principal de Apache Spark. Esta clase se incluyó en la versión 2.0 de Apache Spark con el objetivo de hacer más sencillo el uso a los nuevos usuarios y para permitir dar estructura a los datos con los que se está trabajando [Damji2016].

Los Dataframe son datasets que se organizan idealmente en columnas con nombre. Se pueden construir a partir de una variedad de fuentes diferentes, como archivos de datos estructurados, bases de datos externas o RDD existentes.

Los Dataframe se utilizan principalmente para procesar una gran cantidad de datos estructurados, siendo más potentes que los RDD en este aspecto. Tienen varias características que comunes con los RDD, como la capacidad de computación distribuida, la inmutabilidad, el trabajar en memoria y el ser tolerantes a fallos e incorporan nuevas funcionalidades como permitir un manejo personalizado de la memoria, un optimizador de entrada de datos y mejor integración con otras herramientas Big Data. Además, proporcionan un nivel de abstracción superior.

Sobre la clase Dataframe también se realizan operaciones similares a las transformaciones y acciones, siendo los nombres de las mismas parecidas a comandos en SQL, como se puede ver en la tabla 3.3. Todas las operaciones posibles sobre los RDD también funcionan sobre los Dataframe.

Operación	Resultado
select(*columna*)	Equivalente al comando Select(*columna*) en SQL.
filter(*condición*)	Equivalente al comando Where(*condición*) en SQL.
groupby(*columna*)	Equivalente al comando Groupby (*columna*) en SQL.
printschema()	Devuelve en texto un esquema de las columnas del Dataframe y sus propiedades.
withcolumn(*nuevaC, función*)	Crea un nuevo Dataframe que contiene la columna *nuevaC* como resultado de ejecutar *función* sobre los elementos del Dataframe.

Tabla 3.3 Ejemplos de operaciones para la clase Dataframe

Fuente: Elaboración propia

3.6 Aplicaciones en Apache Spark

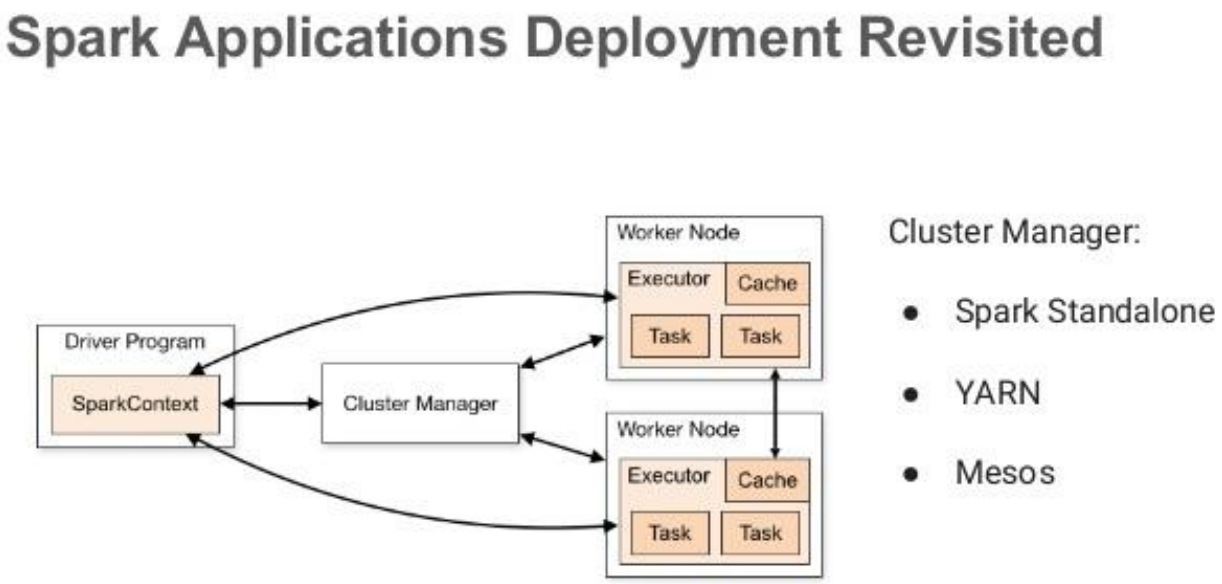

Figura 3.6 Arquitectura de una aplicación en Apache Spark

Fuente: Documentación de Apache Spark

Apache Spark utiliza una arquitectura master-slave que consta de un programa driver, que se ejecuta como un nodo master, y muchos ejecutores que se ejecutan en los nodos de trabajo en el clúster [IntelliPaat2017, Or2015]. A continuación se describen los elementos que componen la arquitectura:

Programa driver

El programa driver es el proceso donde se ejecuta la función Main de una aplicación de Apache Spark. En él se encuentra el código escrito por el desarrollador donde se crea un SparkContext que permite configurar el entorno de ejecución, se trabaja con los RDD y Dataframes y se realizan trasformaciones y acciones.

El programa driver lee el código de la aplicación y lo divide en tareas que planifica y distribuye a los ejecutores. Para ello crea un Grafo Dirigido Acíclico (Figura 3.7) en el que se representan las operaciones que se van a realizar y su orden.

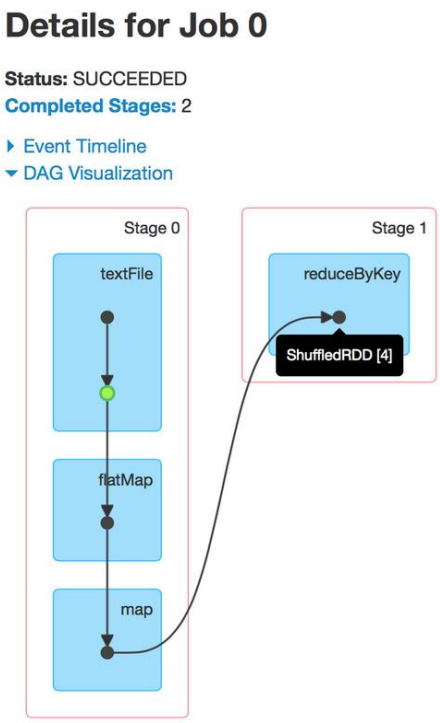

Figura 3.7 Ejemplo de Grafo Dirigido Acíclico

Fuente: databricks.com

Ejecutores

Las tareas individuales en los trabajos de Apache Spark se ejecutan en los ejecutores. Los ejecutores se inician al comienzo de la aplicación Apache Spark y luego se ejecutan durante toda la vida útil de una aplicación. Si un ejecutor de falla, Apache Spark está diseñado para que la aplicación pueda continuar sin problemas.

Los ejecutores tienen dos roles:

- Ejecutar las tareas que compone la aplicación y devolver el resultado al programa driver.
- Proporcionar almacenamiento en memoria para los RDD que el usuario almacena en caché.

En la figura Figura 3.7 se puede ver un ejemplo de cómo se distribuyen las tareas a los ejecutores en una aplicación de Apache Spark y el tiempo que tardan.

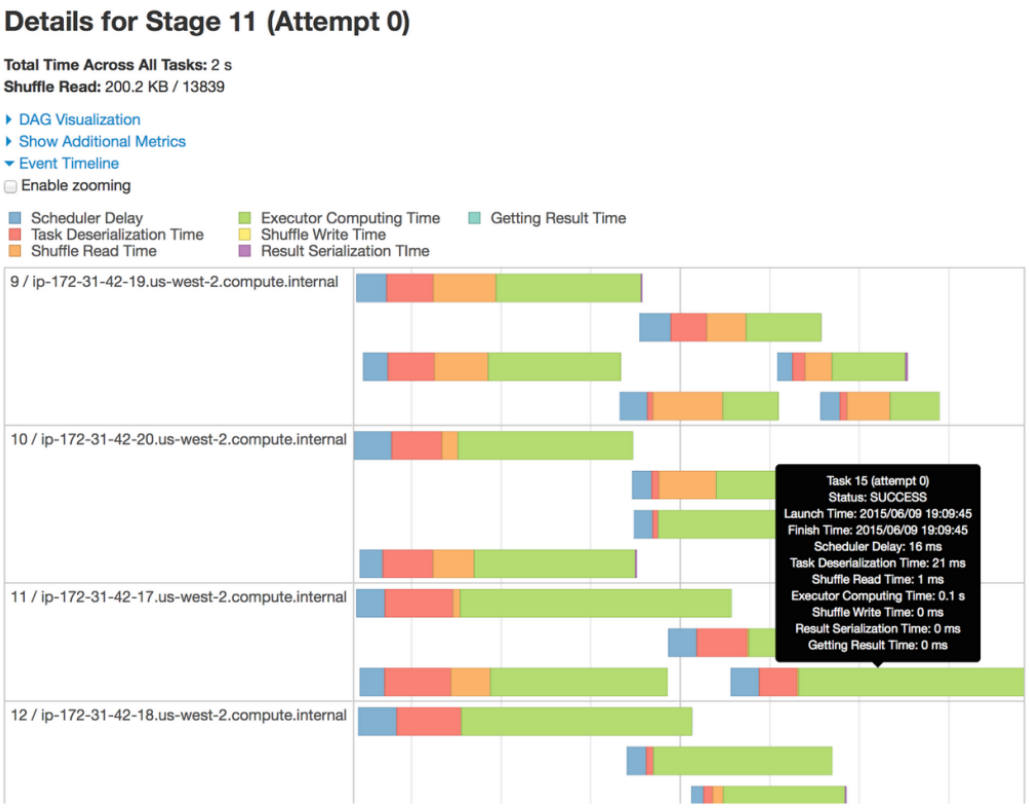

Figura 3.7 Vista de la distribución de las tareas a los ejecutores en una aplicación de Apache Spark

Fuente: databricks.com

Clúster manager

El clúster manager es el encargado de lanzar los ejecutores y dar los recursos necesarios al programa driver según se especifica en el SparkContext. También se encarga de controlar la ejecución de las tareas en los ejecutores.

Apache Spark admite distintos clúster, y el clúster a utilizar depende de las preferencias del desarrollador.

3.7 Apache Spark MLlib

MLlib es la biblioteca de *machine learning* de Apache Spark como se ha visto anteriormente.

Provee de funcionalidades de alto nivel como:

- Algoritmos de *machine learning* para clasificación, regresión, clustering y filtrado colaborativo.
- Definición de características: extracción de atributos, trasformaciones, reducción de dimensionalidad y selección de atributos.
- Pipelines: herramientas para construir, testear y ajustar pipelines de *machine learning.*
- Persistencia: facilita el guardado y la carga de algoritmos, modelos y pipelines.
- Utilidades de algebra lineal, estadística y manejo de datos.

MLlib está dividida en dos paquetes: *spark.ml* y *spark.mllib*.

- spark.ml es la API basada en la clase Dataframe. Esta hace más fácil la construcción de pipelines para procesos de aprendizaje máquina y permite trabajar con ella de manera uniforme entre los distintos lenguajes de programación.
- spark.mllib es la API basada en los RDD (Resilient Distributed Dataset). Es la más antigua de las dos y entró en modo de mantenimiento (solo recibiendo arreglos a bugs) para centrarse en el desarrollo de spark.ml e igualar sus funcionalidades.

Actualmente los dos paquetes ya han alcanzado paridad entre sus funcionalidades, pero spark.mllib seguirá en modo de mantenimiento y las nuevas funcionalidades que se desarrollen se incluirán solo en spark.ml.

4. DESCRIPCIÓN DE LOS ALGORITMOS A DESARROLLAR

En esta sección se detallan los dos algoritmos de ensembles que se han desarrollado: *Stacking* y *Bagging*.

El objetivo de estos algoritmos es combinar varios clasificadores individuales para obtener mejores resultados que los obtenidos mediante la ejecución de los mismos por separado.

Para ello se pueden incluir varios tipos de clasificadores individuales y con distintos parámetros en los ensembles.

4.1 Stacking

Stacking es un algoritmo de ensembles que consiste en entrenar varios modelos base con un dataset y utilizar las predicciones resultantes de esos modelos para entrenar otro nuevo modelo que realizará las predicciones finales. Con este modelo, llamado meta-learner, se busca resaltar los puntos fuertes de los modelos anteriores para obtener mejores resultados que ellos. Se puede utilizar cualquier modelo de *machine learning* tanto para los modelos base como para el modelo meta-learner. En la Figura 4.1 se poder ver un esquema del funcionamiento de Stacking.

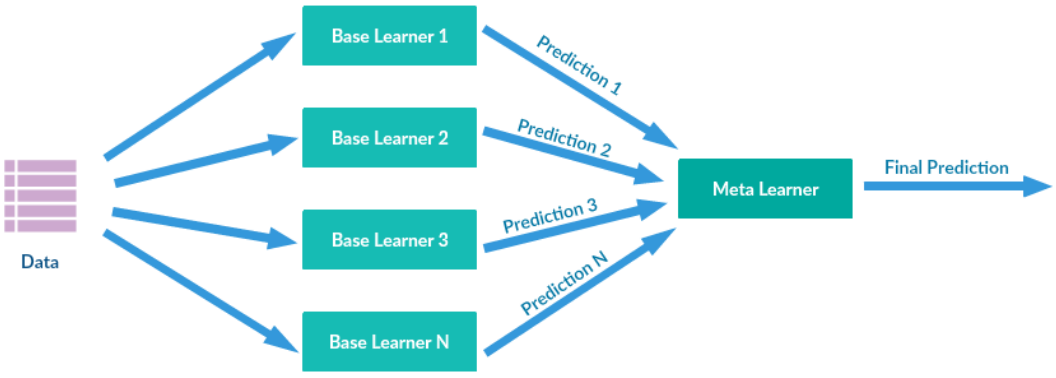

Figura 4.1 Esquema de Stacking

Fuente: medium.com

El funcionamiento de este algoritmo se divide en varios pasos:

Paso 1:
Se divide el dataset en dos conjuntos, entrenamiento y test. A continuación, el conjunto de entrenamiento se divide en N partes. (Figura 4.1.1)

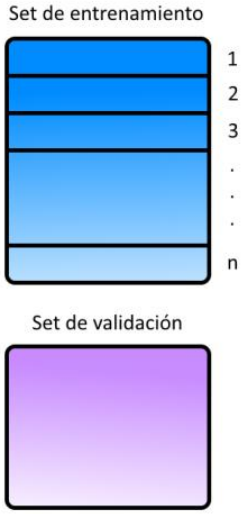

Figura 4.1.1 Paso 1 de Stacking
Fuente: Elaboración propia

Paso 2:
Se elige un modelo base de tipo A.

Después se crea un modelo de ese tipo elegido que se entrena con el conjunto resultante de unir las N partes del conjunto de entrenamiento menos una y se realizan predicciones para esa parte restante. Esto se realiza con todos los conjuntos posibles con modelos del mismo tipo que el inicial. Al unirse todas las predicciones en un solo conjunto se obtiene el conjunto de predicciones sobre el conjunto de entrenamiento. (Figura 4.1.2)

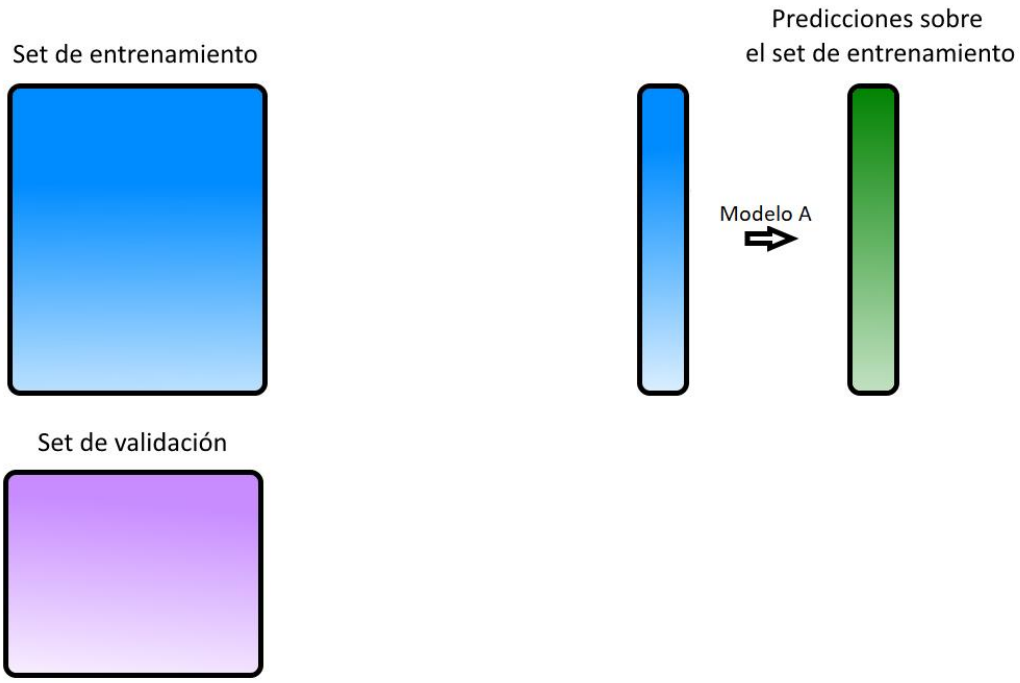

Figura 4.1.2 Paso 2 de Stacking
Fuente: Elaboración propia

Paso 3:
Se crea un nuevo modelo del mismo tipo que el modelo base elegido (tipo A), que se entrena con el conjunto de predicciones sobre el conjunto de entrenamiento resultante del paso 2.

Paso 4:
Con el modelo creado en el paso 3, se realizan predicciones sobre el conjunto de test, resultando en un conjunto de predicciones sobre el conjunto de test. (Figura 4.1.3)

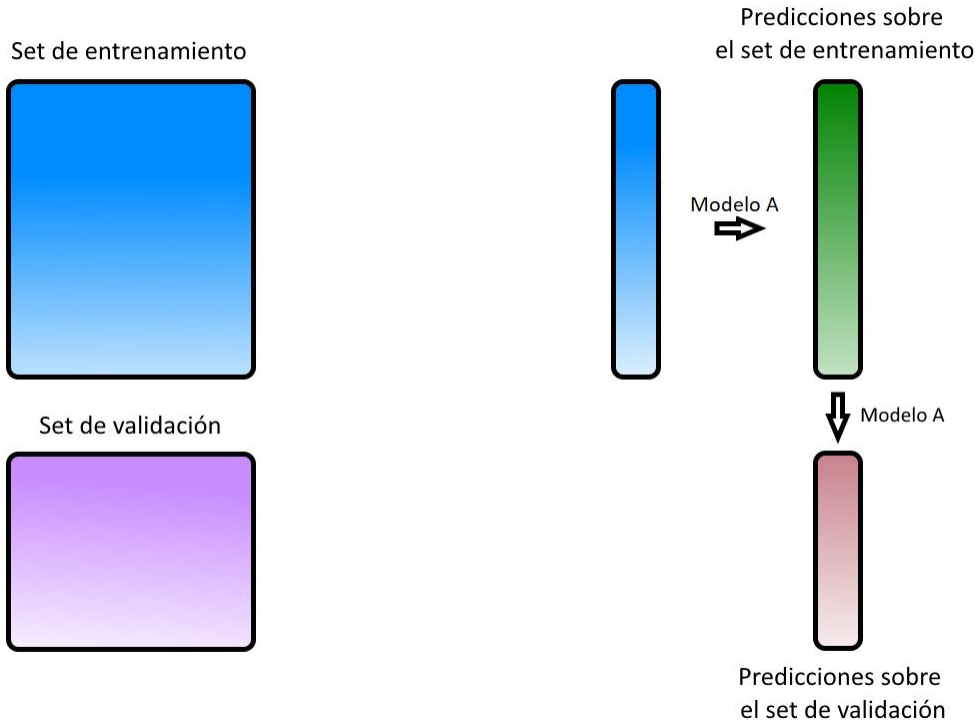

Figura 4.1.3 Paso 4 de Stacking
Fuente: Elaboración propia

Paso 5:
Se repiten los pasos 2 a 4 para los demás modelos base elegidos, resultando en varios conjuntos de predicciones sobre los conjuntos de entrenamiento y test. Los pasos para cada modelo base son independientes unos de otros y pueden ejecutarse en paralelo. (Figura 4.1.4)

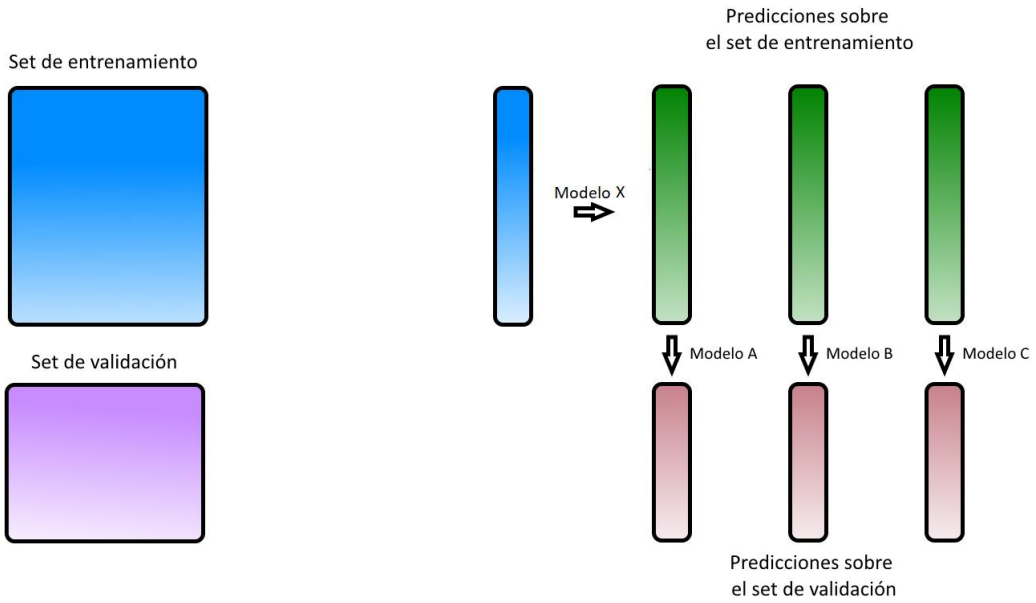

Figura 4.1.4 Paso 5 de Stacking
Fuente: Elaboración propia

Paso 6:
Se utilizan las predicciones obtenidas sobre el conjunto de entrenamiento para construir un nuevo modelo. (Figura 4.1.5)

Paso 7:
El nuevo modelo se utiliza para realizar las predicciones finales sobre el conjunto de test. (Figura 4.1.5)

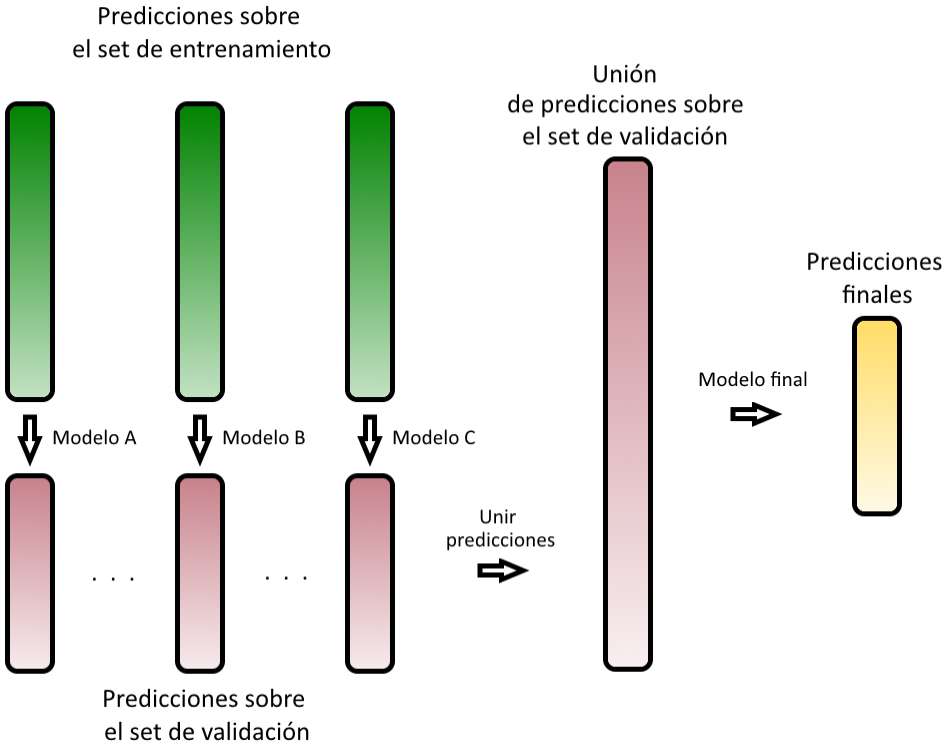

Figura 4.1.5 Paso 7 de Stacking
Fuente: Elaboración propia

4.2 Bagging

Bagging (abreviación de Bootstrap Aggregating) es un algoritmo de ensembles que consiste en combinar los resultados de varios modelos para obtener resultados más generalizados y reducir la varianza.

En la Figura 4.2 se poder ver un esquema del funcionamiento de Bagging.

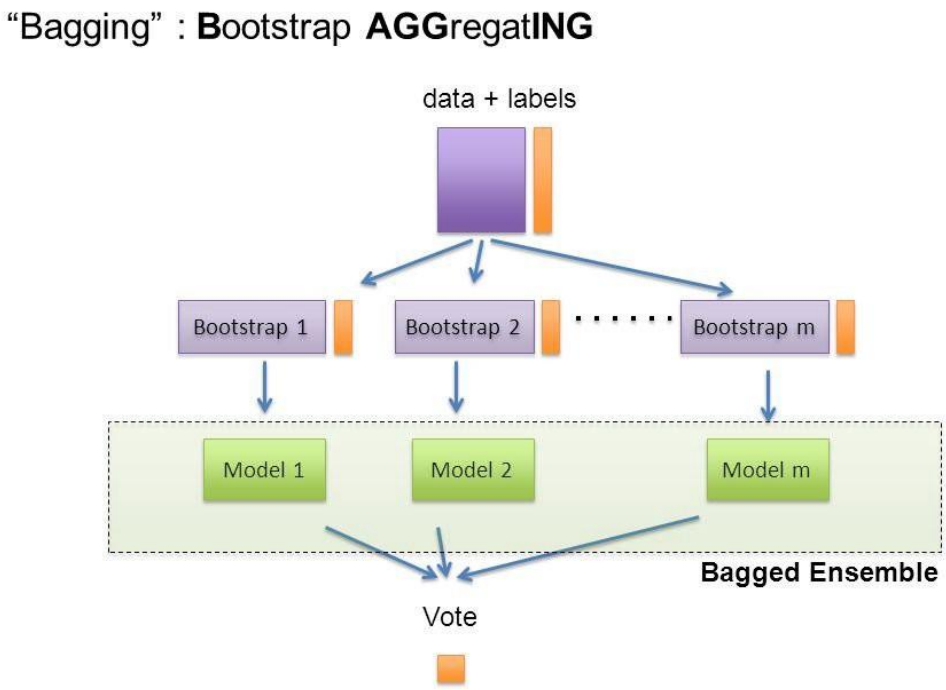

Figura 4.2 Esquema de Bagging

Fuente: medium.com

El funcionamiento de este algoritmo se divide en varios pasos:

Paso 1:
Se divide el dataset en dos, entrenamiento y test. (Figura 4.2.1)

Figura 4.2.1 Paso 1 de Bagging
Fuente: Elaboración propia

Paso 2:
Se crean varios subconjuntos del dataset original, con reemplazamiento y sin escoger necesariamente todos los atributos. (Figura 4.2.2)

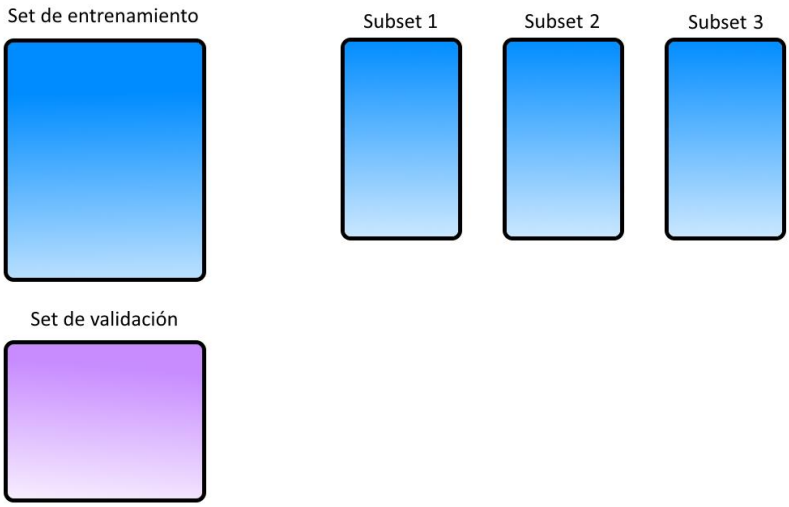

Figura 4.2.2 Paso 2 de Bagging
Fuente: Elaboración propia

Paso 3:

Se crea un modelo a partir de cada uno de esos subconjuntos y se obtienen las predicciones sobre los mismos. Los modelos son completamente independientes unos de otros y por lo tanto pueden ejecutarse en paralelo. (Figura 4.2.3)

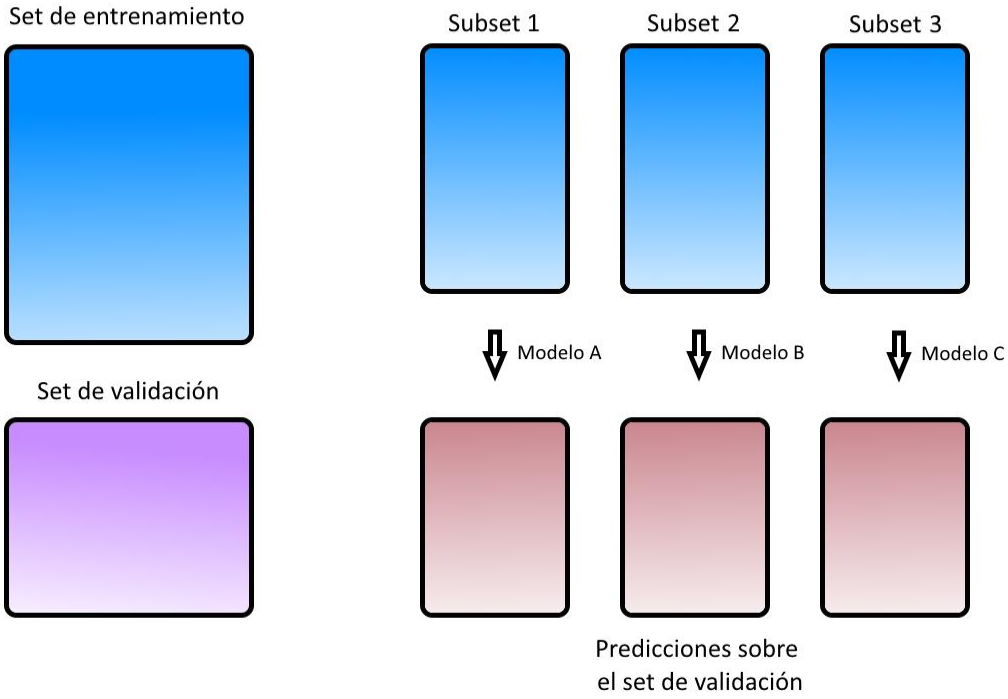

Figura 4.2.3 Paso 3 de Bagging

Fuente: Elaboración propia

Paso 4:

Se combinan las predicciones de los modelos para obtener los resultados finales. Para combinar los resultados y obtener las predicciones finales se pueden utilizar distintos métodos combinatorios como quedarse con la predicción más votada o hacer una media o una media ponderada de todas las predicciones. (Figura 4.2.4)

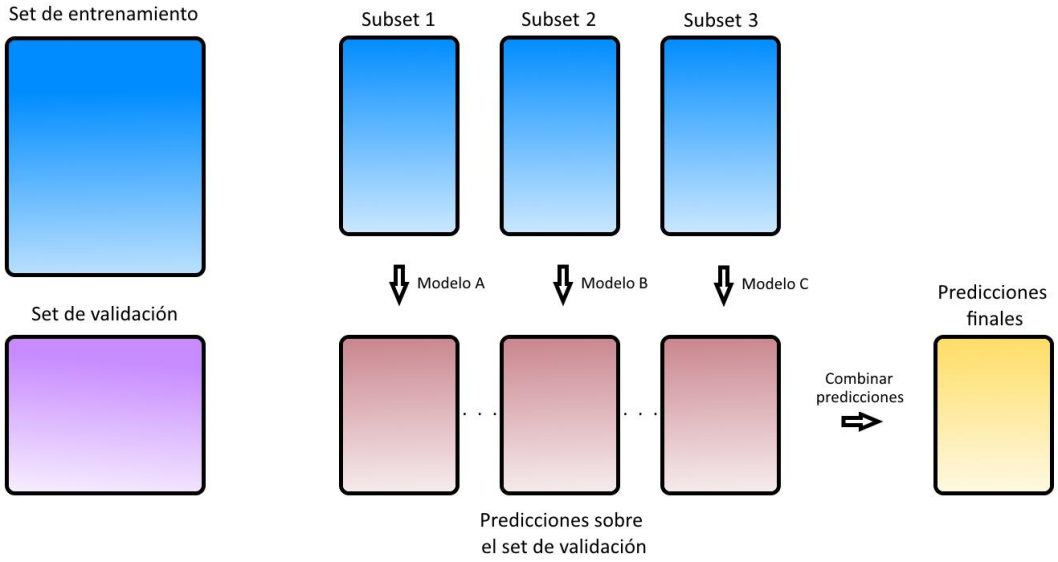

Figura 4.2.4 Paso 4 de Bagging

Fuente: Elaboración propia

5. DISEÑO E IMPLEMENTACIÓN

En esta sección se presentan las fases de diseño e implementación que se ha llevado a cabo para los ensembles Stacking y Bagging.

5.1 Diseño

Para el desarrollo de este trabajo se ha utilizado la biblioteca de *machine learning* MLlib de Apache Spark.

Esta biblioteca contiene implementaciones de los clasificadores de Regresión Logística, Decision Tree y Naive Bayes que se van a utilizar para la ejecución de los ensembles y evaluaciones de rendimiento para cada uno de ellos.

MLlib permite configurar los clasificadores de manera que se les pueden pasar distintas parámetros de ejecución para que se adecúen a los datasets sobre los que se van a utilizar y a lo que busque el desarrollador.

Habiendo probado las dos APIs spark.ml y spark.mllib, y puesto que las dos tienen las funcionalidades de los clasificadores que necesito implementadas, me he decantado por usar spark.mllib por haber encontrado más fácil de trabajar con RDD para los ensembles a desarrollar.

Para trabajar con los clasificadores con la API de spark.mllib es necesario que los objetos dentro los RDD con los que se va a trabajar sean de la clase *LabeledPoint*. Cada *LabeledPoint* dentro de un RDD corresponde a un elemento de los dataset. Un *LabeledPoint* está compuesto por un *Double*, que representa la clase del elemento al que corresponde, y un array de *Double* que representa los atributos de ese elemento.

Por lo tanto, para trabajar con un dataset primero será necesario transformarlo a un RDD de *LabeledPoint*.

Si los atributos del dataset son nominales habrá que transformarlos a números. Para ello se introducen todos los valores nominales posibles en un array, la posición de cada valor nominal en el array indica su número equivalente.

También hay que tener en cuenta que para utilizar el clasificador de Naive Bayes no pueden existir valores negativos en los atributos del dataset, por lo que si hay algún atributo con valores negativos, este se normalizará.

Para realizar estas funciones se han creado las clases Dataset y Attribute:

- La clase Dataset se encarga de cargar los datasets, contiene las instancias de los mismos y la información sobre sus atributos en un array de Attribute. También se encarga de transformar los atributos nominales a números y normalizarlos.
- La clase Attribute contiene la información sobre un atributo: su nombre, su tipo, un array que contiene los valores nominales que puede tomar (para transformarlos a números) y los valores mayor y menor que toma (para la normalización).

5.2 Clases implementadas

Para la creación de los modelos se ha implementado las clases *ModeloLR*, *ModeloDT* y *ModeloNaiveBayes*, cada una con dos funciones:

- Una función llamada *Modelo* a la que se le pasa un dataset y los argumentos que se prefieren para la creación del modelo (distintos según si son Regresión Logística, Decision Tree o Naive Bayes).
- Una función llamada *precisionModelo*, a la que se le pasa como argumentos un modelo creado con la función anterior y un dataset de test y devolverá la tasa de clasificación que obtiene el modelo.

Para la implementación de Bagging y Stacking se han creado la clase *MainBagging* y *MainStacking*. Estas clases hacen uso de la clase Dataset para cargar el dataset que va a utilizar y después ejecutan el ensemble con los modelos que se pasan como argumento al inicio del programa.

Dentro de la clases *MainBagging* y *MainStacking* se ha creado la función *SubsetModelo*. Esta función se encarga de crear cada modelo por separado y devuelve un objeto de tipo *Future* que contiene un RDD de Double, que son las predicciones del modelo. El uso del objeto *Future* se utiliza para paralelizar la ejecución de los modelos, ya que un objeto *Future* actúa como un *placeholder* para un objeto que todavía no se ha creado.

Al iniciarse la ejecución de Bagging se crea un array de *Future* que contienen RDD de *Double*, y una vez están todos creados, se comienza la ejecución de los mismos. A cada objeto *Future* le corresponderá la ejecución de un modelo, y el RDD de double que devolverá serán las predicciones resultantes de ese modelo. Una vez haya finalizado la ejecución de un *Future* se guardará el RDD de Double resultante en un array. Cuando haya finalizado la ejecución de todos los *Future*, significará que se ha terminado de ejecutar todos los modelos y se puede proseguir con la ejecución del ensemble.

Para la ejecución de Stacking se han utilizado los *Future* de manera distinta que Bagging. Dentro de la función *SubsetModelo* no se han creado directamente los modelos, a diferencia de Bagging. Para Stacking se ha creado la clase *StackingSubpaso* para realizar los pasos 2 a 4 con los distintos modelos, y la función *SubsetModelo*llama a la función *Stacking* de la clase *StackingSubpaso* con los argumentos del modelo que quiere crear para que esta realice los pasos 2 a 4.

Así, dentro de la función *Stacking* de *StackingSubpaso:*

- Se crean los N modelos del paso 2 y se unen sus predicciones.
- Se crea el modelo del paso 3 con la unión de predicciones anterior.
- Se realizan predicciones con el modelo del paso 3 sobre el conjunto de test para obtener un conjunto de predicciones sobre el mismo (Paso 4).
- El conjunto de predicciones obtenido será el objeto RDD de *Double* que devolverá el objeto *Future*.

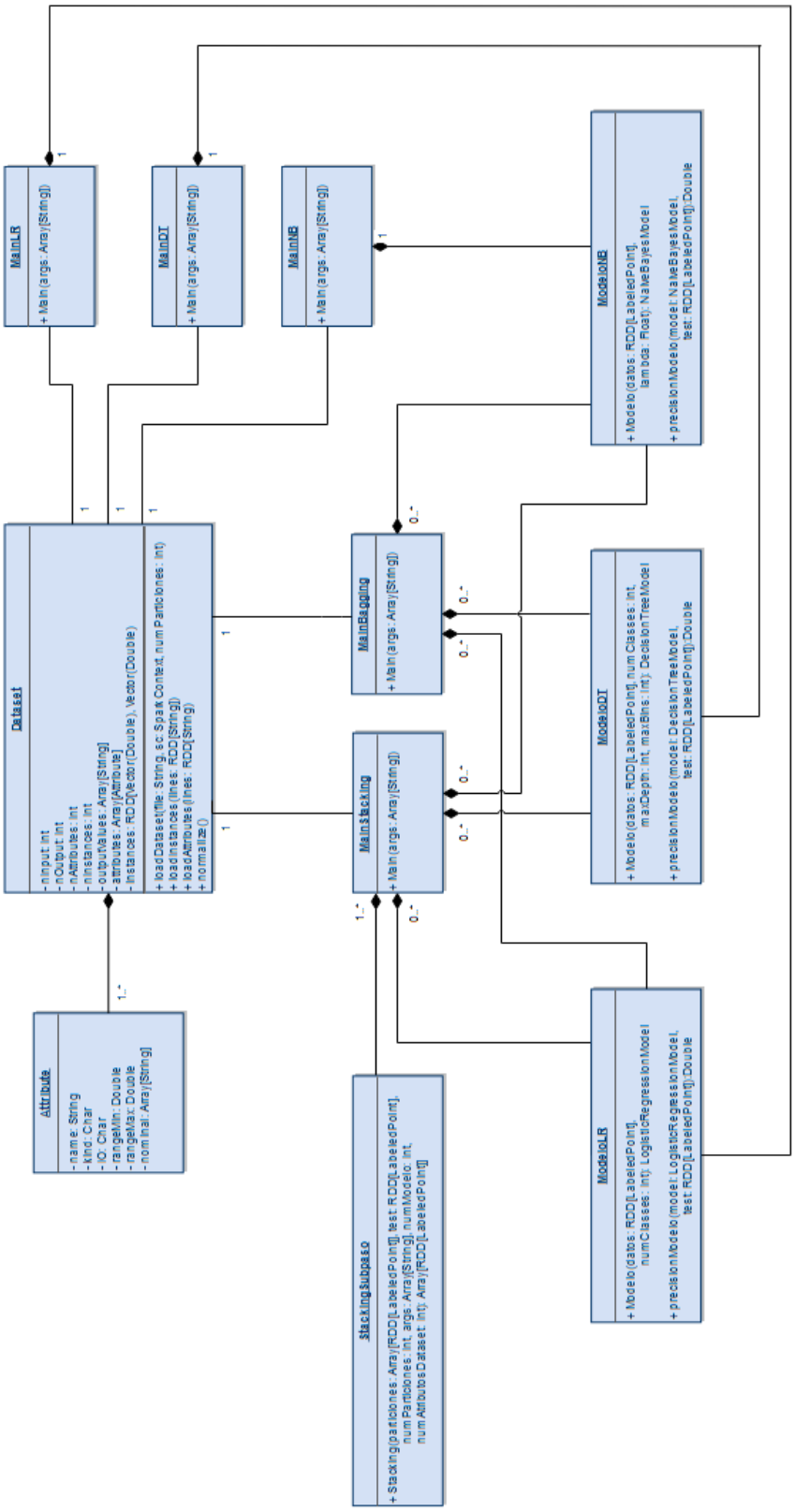

Figura 5.1 Diagrama de clases de los ensembles y los modelos

Fuente: Elaboración propia

Para la creación de las clases que contienen los modelos es necesario importar múltiples bibliotecas de MLlib. En la Figura 5.2 se pueden ver las bibliotecas utilizadas que para la creación de las clases ModeloLR, ModeloDT y ModeloNaiveBayes.

También se ha hecho uso de la biblioteca *regression* con *LabeledPoint* ya que es necesaria para trabajar con los RDD de LabeledPoint, que son los que utiliza Spark.mllib.

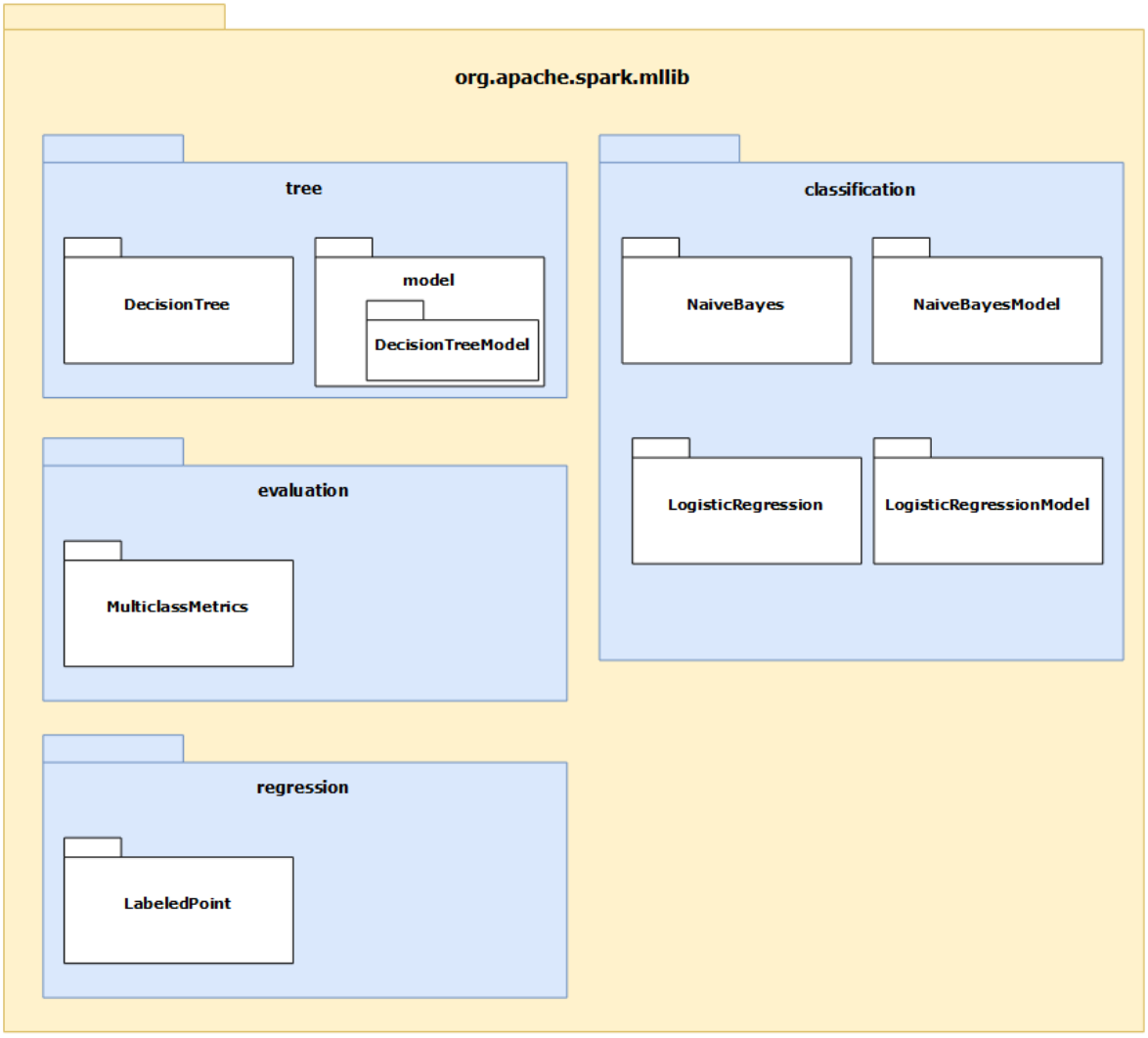

Figura 5.2 Bibliotecas de MLlib utilizadas en el proyecto

Fuente: Elaboración propia

5.3 Lenguaje de implementación

Apache Spark proporciona múltiple APIs para los lenguajes de programación Java, Python, Scala y R, aunque está principalmente desarrollado en Scala y la API de este lenguaje es la que recibe primero las actualizaciones y las nuevas funcionalidades.

De los lenguajes ofrecidos por Apache Spark se ha elegido Scala para la realización de este trabajo por ser el lenguaje más eficiente en Apache Spark y por su mayor eficiencia para aplicaciones concurrentes.

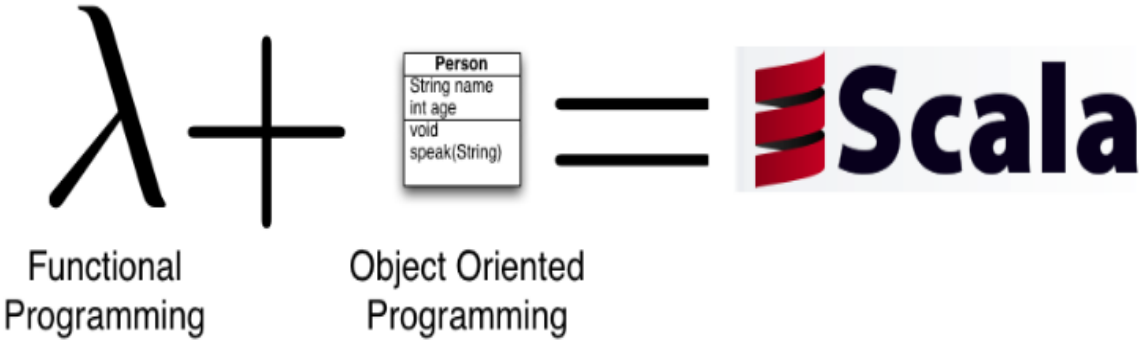

Figura 5.3 Scala

Scala es un lenguaje de programación multiparadigma ya que combina programación funcional con programación orientada a objetos. Corre sobre la máquina virtual de Java (JVM) por lo que es compatible con bibliotecas y aplicaciones escritas en Java [Alexander2013].

Algunas características destacables de Scala son:

- Inferencia de tipos: en Scala no es necesario mencionar explícitamente los tipos de datos que devuelve una función. Scala es capaz de deducir el tipo de dato a devolver mediante la última expresión presente en la función.
- Computación *lazy:* en Scala es posible declarar expresiones *lazy* para que solo se computen cuan sea requeridas.
- Gran flexibilidad sintáctica: lo que permite reducir el número de líneas que se escriben, especialmente comparado con Java.

- La clase Trait: un Trait es parecido a una implementación parcial de la clase Interface de Java. Permite ser teniendo todos los métodos abstractos o solo algunos de ellos.
- Inmutabilidad: Scala obliga a distinguir entre las variables y objetos inmutables (cuyo valor no puede cambiar) y los mutables.
- Un rico sistema de tipos que permite hacer un uso más seguro de las clases abstractas.

6. ANÁLISIS DE RESULTADOS

En esta sección se exponen los resultados de los experimentos que se han llevado a cabo con el objetivo de validar los ensembles implementados y las ventajas y desventajas de su utilización.

Para ello se han realizado primero ejecuciones por separado de cada uno de los clasificadores de MLlib que ha han utilizado, para comparar después los resultados obtenidos cuando estos se incluyen en los ensembles.

6.1 Características de la experimentación

Para la experimentación se han utilizado datasets de clasificación estándar públicos obtenidos del repositorio Keel.

En la tabla 6.1 se muestran los tres datasets utilizados, con el número de ejemplos que contienen, el número de atributos y su tipo y el número de clases.

Dataset	Nº de ejemplos	Atributos Reales	Atributos Enteros	Atributos Nominales	Nº de clases
Magic	19020	10	0	0	2
Shuttle	58000	0	9	0	7
Poker	1025010	0	10	0	10

Tabla 6.1 Características de los datasets
Fuente: Elaboración propia

A continuación, se hace una descripción de dichos datasets:

Poker: este dataset contiene ejemplos que representan manos de póker de 5 cartas sacadas de una baraja de 52 cartas. Cada carta se describe utilizando dos atributos que representan el palo y el rango, sumando en total 10 atributos. La clase representa la mano de póker obtenida con las 5 cartas.

Shuttle: este dataset contiene ejemplos que indican según las condiciones en las que se encuentre una nave espacial cómo debería realizarse el aterrizaje. Los atributos

indican las condiciones de la nave y las clases si el aterrizaje debería ser manual o automático y hasta qué punto.

Magic: este dataset contiene datos generados para simular el registro de partículas gamma en un telescopio atmosférico Cherenkov y discernir su tipo según las imágenes que generan. Los atributos indican las propiedades de las partículas y la clase su tipo.

Los experimentos realizados con los datasets se han realizado utilizando validación cruzada k-fold. Esto permite que los resultados sean independientes de la partición que se escoja. Para la validación cruzada se han utilizado la versión de 5 particiones realizadas con k-fold para cada dataset que proporciona el repositorio Keel.

Para cada partición de validación cruzada de los datasets se han realizado 4 ejecuciones, utilizando 1, 2, 4 y 8 particiones internas en RDD, de forma que se pueda comprobar cómo escalan los ensembles al paralelizar su ejecución.

Para la ejecución de los ensembles es necesario especificar como parámetro el tipo de los clasificadores que se van a utilizar en el mismo. Para cada ejecución se especifican los clasificadores de MLlib que se le han pasado al ensemble y su configuración.

Como medida de calidad de los modelos se utilizará la tasa de clasificación que se define como el número total de aciertos dividido entre el número total de ejemplos del conjunto de test.

Las ejecuciones se han realizado en el clúster del Centro de Estudios Avanzados en Tecnologías de la Información y de la Comunicación (CEATIC) de la Universidad de Jaén, con las siguientes características:

- 16 nodos de computación para Big Data repartidos en 4 servidores Bull NovaScale R434-F3 que ofrecen un total de 310 núcleos a 2.5Ghz.
- 1TB de memoria RAM a 1866Mhz.

6.2 Experimentos para Stacking

Las siguientes tablas muestran los resultados de las tasas de clasificación de las ejecuciones de Stacking comparado con un clasificador en solitario. Cada tabla muestra la comparación con un clasificador distinto.

Para la ejecución de Stacking se le han pasado como parámetros 10 clasificadores de un tipo y para el modelo final otro clasificador del mismo tipo. A todos los clasificadores se les han pasado los mismos parámetros de ejecución adaptados para cada dataset.

6.2.1 Resultados de tasas de clasificación

Dataset	Clasificador en solitario LR	Stacking LR	Diferencia
Magic	78.40%	78.35%	-0.05
Shuttle	91.80%	92.00%	+0.20
Poker	50.00%	50.00%	0.00

Tabla 6.2.1 Tasas de clasificación para Regresión Logística

En la tabla 6.2.1 se puede observar que el clasificador en solitario y Stacking dan resultados muy similares para la tasa de clasificación con Regresión Logística, siendo mejores para Stacking con en el dataset Shuttle.

Dataset	Clasificador en solitario DT	Stacking DT	Diferencia
Magic	83.60%	83.70%	+0.10
Shuttle	99.40%	99.95%	+0.55
Poker	53.40%	55.45%	+2.05

Tabla 6.2.2 Tasas de clasificación para Decision Tree

Con Decision Tree, Stacking sí que consigue mejorar las tasas de clasificación de manera significativa, sobre todo con el dataset Poker donde la tasa de clasificación mejora un 2.05, pero también con el dataset Shuttle, con 0.55, tal y como se pude ver en la tabla 6.2.2.

Dataset	Clasificador en solitario NB	Stacking NB	Diferencia
Magic	66.80%	65.85%	-0.05
Shuttle	78.60%	78.75%	+0.15
Poker	50.00%	50.00%	0.00

Tabla 6.2.3 Tasas de clasificación para Naive Bayes

En la tabla 6.2.3 se puede observar que con Naive Bayes, Stacking obtiene resultados similares de tasas de clasificación al clasificador en solitario para Shuttle y Poker, aunque para el dataset Magic los resultados son un poco peores.

6.2.2 Tiempos de ejecución

En este apartado se va a realizar una comparativa de los tiempos de ejecución de Stacking utilizando como medida el tiempo en segundos.

Nº Nodos	Magic	Shuttle	Poker
1	204.86	406.84	1022.96
2	211.98	415.62	495.26
4	271.74	557.58	385.58
8	315.94	682.40	446.56

Tabla 6.2.4 Tiempos de ejecución en segundos para Regresión Logística

En la tabla 6.2.4 se puede ver que Regresión Logística es el clasificador que más tiempo tarda en ejecutarse de los tres. Para los datasets Magic y Shuttle incrementar el número de nodos a más de 1 hace que los tiempos de ejecución aumenten. Para el dataset Poker, el tiempo de ejecución se reduce si se incrementan los nodos hasta 4, ya que es lo suficientemente grande para mostrar que a medida que aumenta el número de particiones internas, es decir el paralelismo, disminuye el tiempo de ejecución. (Figura 6.2.1)

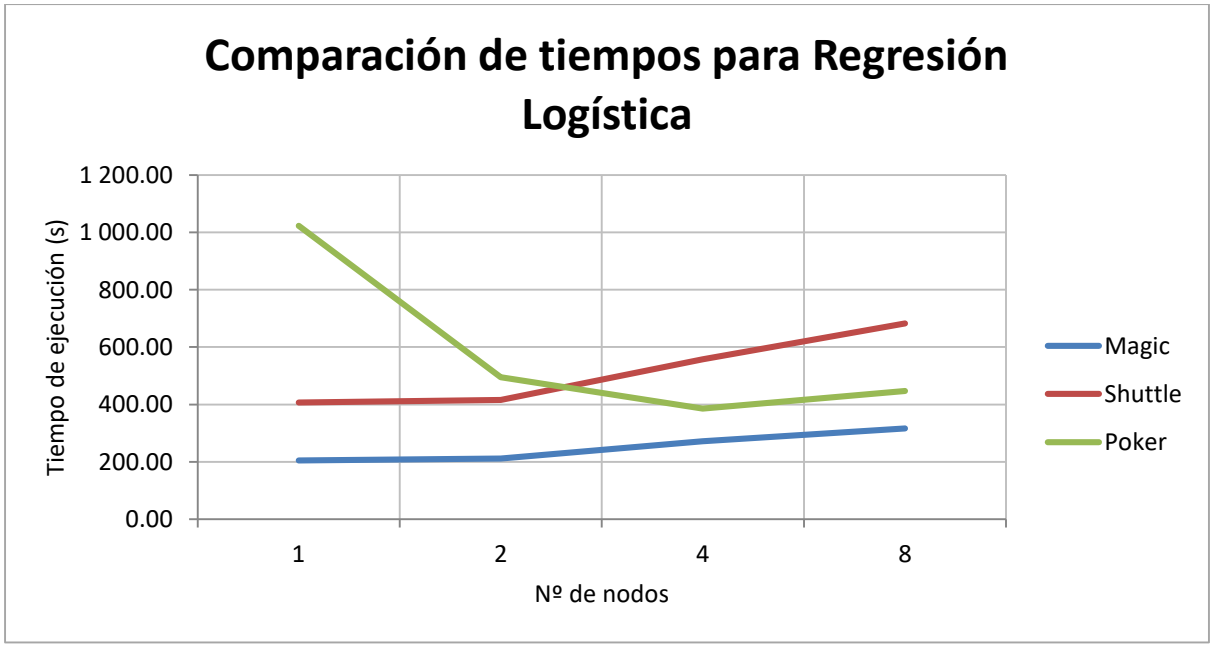

Figura 6.2.1 Tiempos de ejecución en segundos para Regresión Logística

Nº Particiones	Magic	Shuttle	Poker
1	57.44	58.86	84.22
2	68.30	63.06	83.10
4	81.42	73.76	89.56
8	121.10	92.26	109.22

Tabla 6.2.5 Tiempos de ejecución en segundos para Decision Tree

En la tabla 6.2.5 se puede ver que para Decision Tree los tiempos de ejecución son bajos comparados con Regresión Logística y para Magic y Shuttle si se aumentan los nodos aumentan los tiempos de ejecución. Para el dataset Poker los tiempos con 1 y 2 nodos son similares y aumentan para 4 y 8 nodos. (Figura 6.2.2)

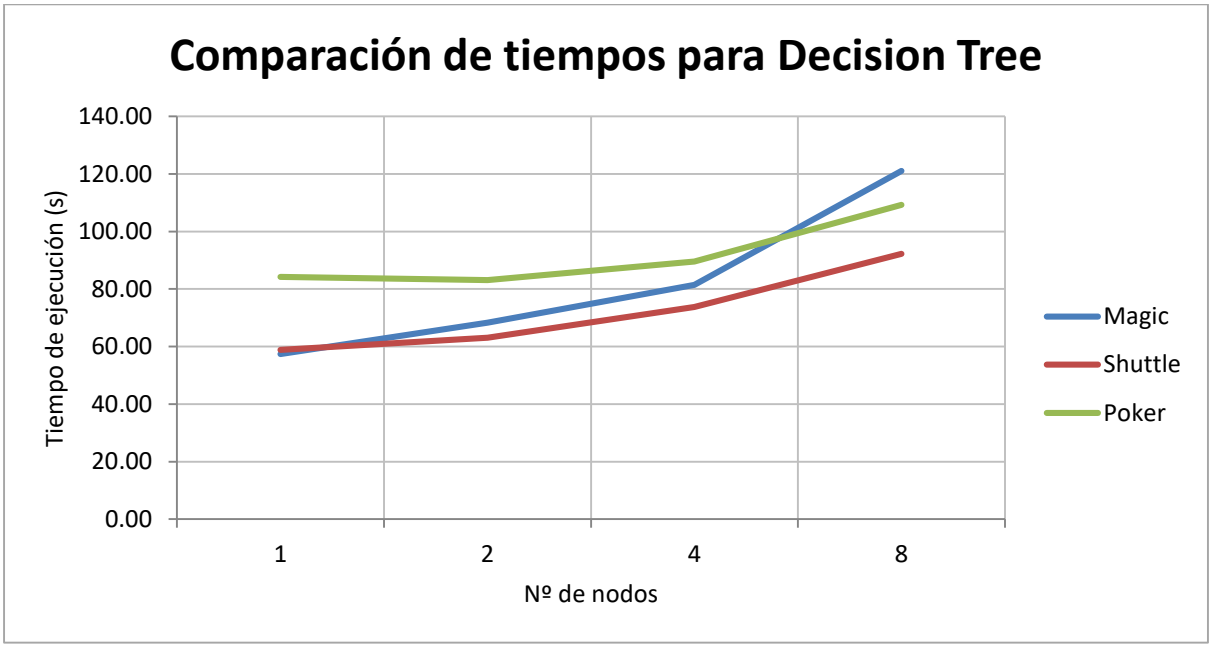

Figura 6.2.2 Tiempos de ejecución en segundos para Decision Tree

Nº Particiones	Magic	Shuttle	Poker
1	21.28	23.02	51.04
2	29.00	31.56	51.38
4	31.46	32.60	43.34
8	35.76	32.42	41.40

Tabla 6.2.6 Tiempos de ejecución en segundos para Naive Bayes

En la tabla 6.2.6 se muestra que los resultados de tiempos de Naive Bayes siguen la misma tendencia que Decision Tree mostrados en la tabla 6.2.5. Si se aumentan los nodos aumentan los tiempos de ejecución. Para el dataset Poker los tiempos con 1 y 2 nodos son similares y disminuyen para 4 y 8 nodos. (Figura 6.2.3)

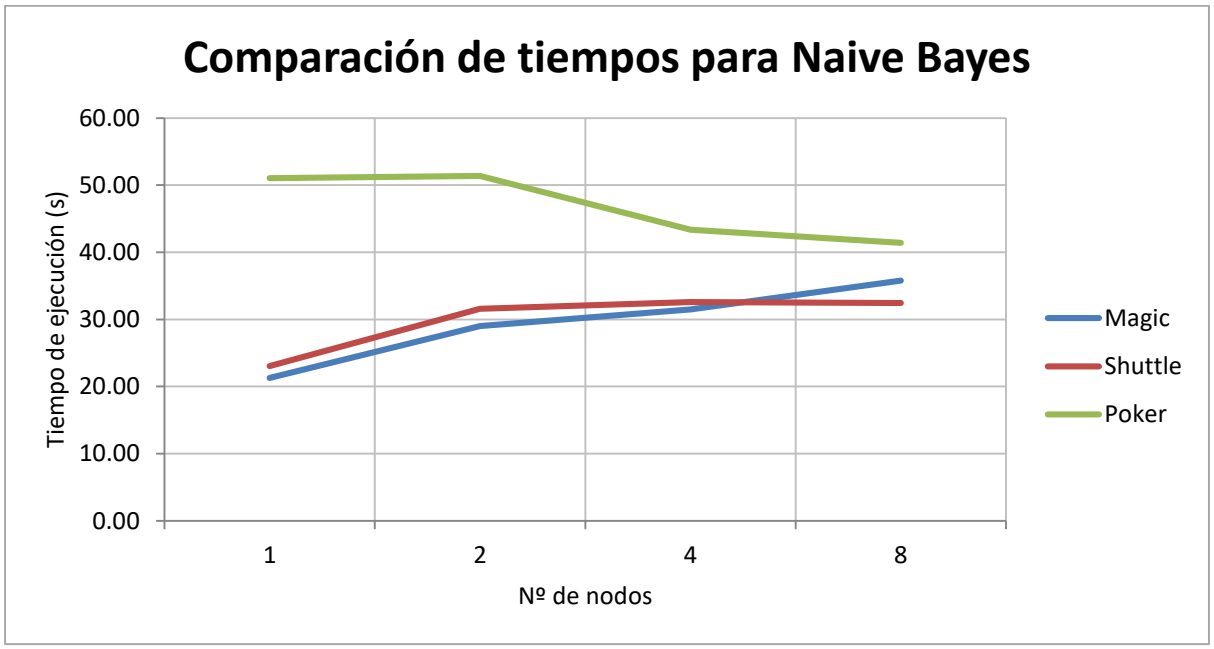

Figura 6.2.3 Tiempos de ejecución en segundos para Naive Bayes

6.3 Experimentos para Bagging

Las siguientes tablas muestran los resultados de las tasas de clasificación de la ejecución de Bagging comparado con un clasificador en solitario. Cada tabla muestra la comparación con un clasificador distinto.

Para la ejecución de Bagging se le han pasado como parámetros 10 clasificadores de un tipo. Para combinar los resultados se ha utilizado el elegir el resultado más frecuente. A todos los clasificadores se les han pasado los mismos parámetros de ejecución adaptados para cada dataset.

6.3.1 Resultados de tasas de clasificación

Dataset	Clasificador en solitario LR	Bagging LR	Diferencia
Magic	78.40%	78.85%	+0.45
Shuttle	91.80%	92.40%	+0.60
Poker	50.00%	50.00%	+-0.00

Tabla 6.3.1 Tasa de clasificación para Regresión Logística

En la tabla 6.3.1 se puede ver que Bagging consigue mejorar un poco las tasas de clasificación para los datasets Magic y Shuttle. No hay diferencia entre los resultados del clasificador y Bagging para Poker.

Dataset	Clasificador en solitario DT	Bagging DT	Diferencia
Magic	83.60%	85.50%	+1.90
Shuttle	99.40%	99.85%	+0.45
Poker	53.40%	57.05%	+3.65

Tabla 6.3.2 Tasa de clasificación para Decision Tree

Con Decision Tree, Bagging consigue mejorar todos los resultados obtenidos por el clasificador en solitario, sobre todo con el dataset Poker, donde la diferencia es de 3.65, como se muestra en la tabla 6.3.2.

Dataset	Clasificador en solitario NB	Bagging NB	Diferencia
Magic	66.80%	73.65%	+6.85
Shuttle	78.60%	78.50%	-0.10
Poker	50.00%	49.45%	-0.55

Tabla 6.3.3 Tasa de clasificación para Naive Bayes

En la tabla 6.3.3 se puede ver que para Naive Bayes Bagging mejora mucho los resultados con Magic. Para Shuttle obtiene resultados similares al clasificador y para Poker empeora un poco.

6.3.2 Tiempos de ejecución

En este apartado se va a realizar una comparativa de los tiempos de ejecución de Bagging utilizando como medida el tiempo en segundos.

Nº Nodos	Magic	Shuttle	Poker
1	25.98	37.76	378.88
2	32.26	41.76	252.10
4	34.00	42.68	208.34
8	43.86	59.94	181.32

Tabla 6.3.4 Tiempos de ejecución en segundos para Regresión Logística

En la tabla 6.3.4 se puede ver que para Regresión Logística los tiempos de ejecución aumentan si se aumenta el número de nodos para Magic y Shuttle. Para Poker si se aumenta el número de nodos el tiempo de ejecución disminuyen al igual que ocurre en Stacking, debido al tamaño del dataset y al incremento de la paralelización. (Figura 6.3.1)

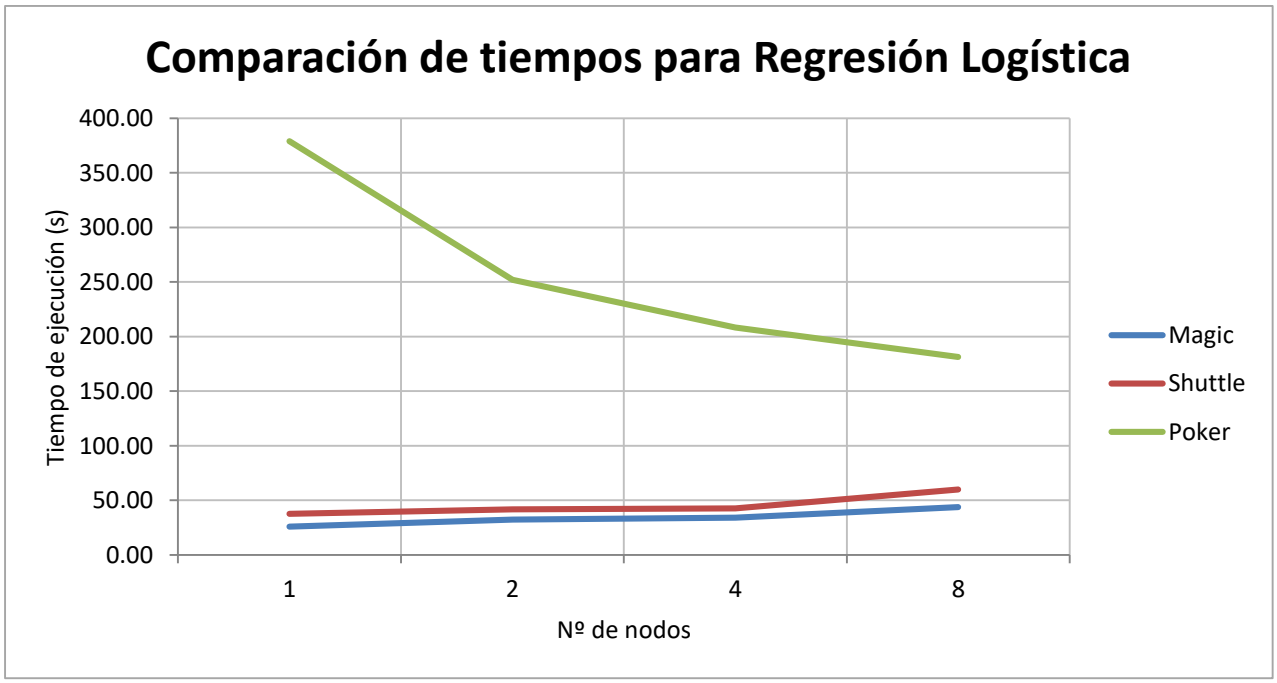

Figura 6.3.1 Tiempos de ejecución en segundos para Regresión Logística

Nº Nodos	Magic	Shuttle	Poker
1	13.14	16.48	218
2	15.56	21.48	205.46
4	19.10	20.48	168.04
8	19.08	21.82	156.98

Tabla 6.3.5 Tiempos de ejecución en segundos para Decision Tree

En la tabla 6.3.5 se muestra que para Decision Tree los tiempos de ejecución también aumentan si se aumenta el número de nodos para Magic y Shuttle, pero no tanto como Regresión Logística. Para Poker también si se aumenta el número de nodos el tiempo de ejecución disminuye, pero no tanto como para Regresión Logística. (Figura 6.3.2)

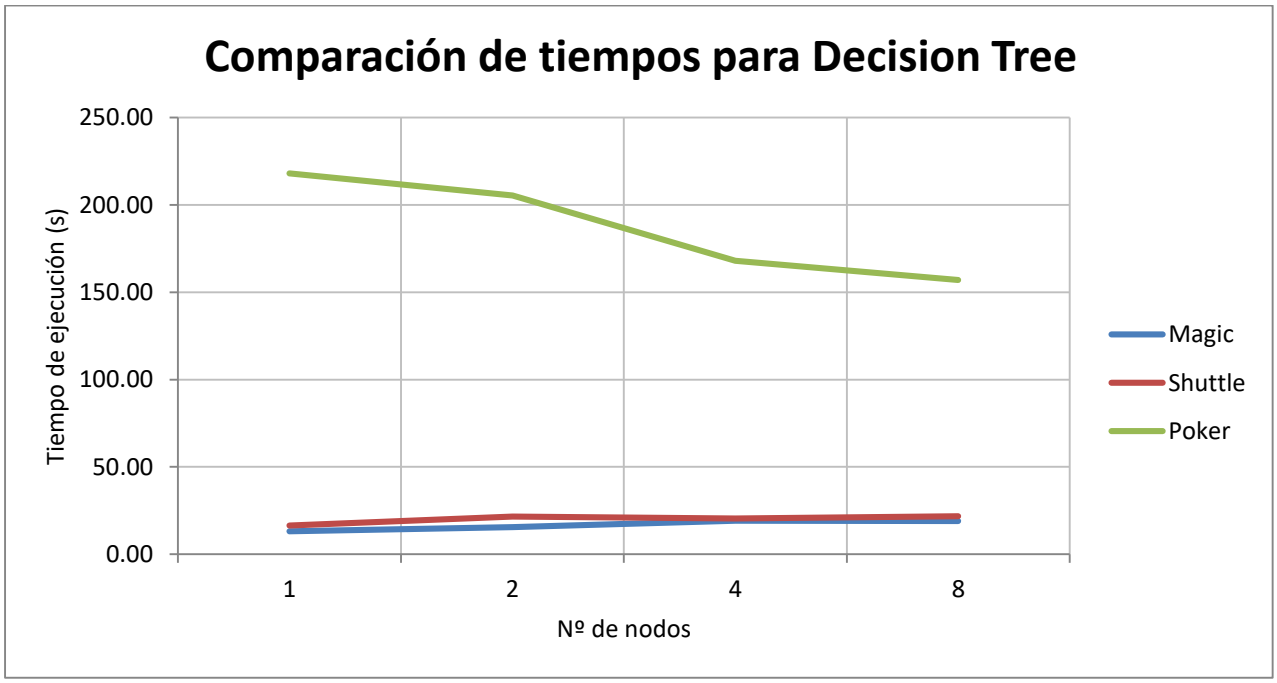

Figura 6.3.2 Tiempos de ejecución en segundos para Decision Tree

Nº Nodos	Magic	Shuttle	Poker
1	13.50	19.44	256.68
2	13.10	27.28	226.00
4	14.02	25.26	176.9
8	14.94	23.54	195.5

Tabla 6.3.6 Tiempos de ejecución en segundos para Naive Bayes

En la tabla 6.3.6 se observa que Naive Bayes tiene unos tiempos parecidos a Decision Tree. Los tiempos de ejecución aumentan un poco si se aumenta el número de nodos para Magic y Shuttle. Para Poker si se aumenta el número de nodos el tiempo de ejecución disminuye hasta 4 nodos y luego aumenta. (Figura 6.3.3)

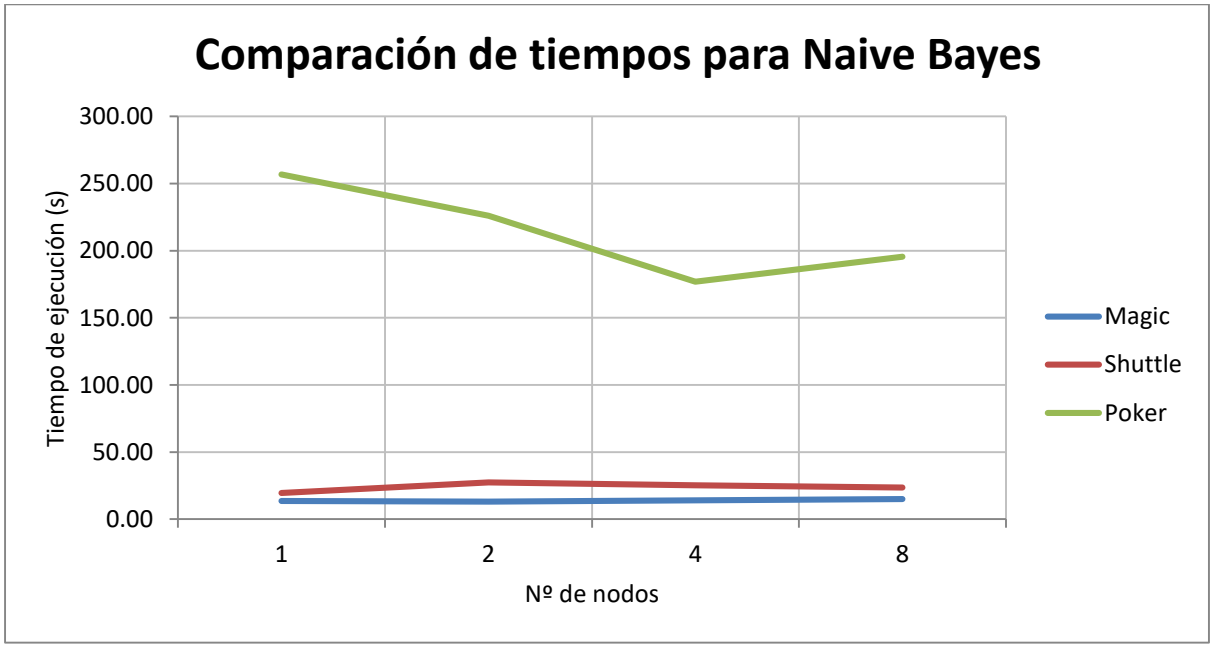

Figura 6.3.3 Tiempos de ejecución en segundos para Naive Bayes

6.4 Comparación de los ensembles

En este apartado se realiza una comparación de todos los resultados mostrados en las tablas de los apartados anteriores, destacándose en las tablas el ensemble que obtiene mejores resultados en negrita.

	Stacking	Bagging	Clasificador en solitario LR
Magic	78.35%	**78.85%**	78.40%
Shuttle	92.00%	**92.40%**	91.80%
Poker	50.00%	50.00%	50.00%

Tabla 6.4.1 Comparación de tasas de clasificación para Regresión Logística

Como se puede ver en la tabla 6.4.1, para Regresión Logística el ensemble que consigue las mejores tasas de clasificación para los datasets Magic y Shuttle es Bagging, siendo estas mejores que Stacking y el clasificador en solitario. Para el dataset Poker los ensembles obtienen el mismo resultado que el clasificador en solitario, sin mejora.

	Stacking	Bagging	Clasificador en solitario DT
Magic	83.70%	**85.50%**	83.60%
Shuttle	**99.95%**	99.85%	99.40%
Poker	55.45%	**57.05%**	53.40%

Tabla 6.4.2 Comparación de tasas de clasificación para Decision Tree

En la comparación de tasas de clasificación mostrada en la tabla 6.4.2 se puede ver, Decision Tree es el clasificador que mejora más con los ensembles, mejorando los resultados para los tres datasets. Bagging consigue los mejores resultados con una diferencia importante para Magic y Poker. Stacking consigue el mejor resultado para Shuttle pero la mejora realizada es más pequeña.

	Stacking	Bagging	Clasificador en solitario NB
Magic	65.85%	**73.65%**	66.80%
Shuttle	**78.75%**	78.50%	78.60%
Poker	50.00%	49.45%	50.00%

Tabla 6.4.3 Comparación de tasas de clasificación para Naive Bayes

Como se muestra en la tabla 6.4.3, Para Naive Bayes, Bagging consigue mejorar mucho los resultados para el dataset Magic. Para Shuttle los ensembles dan resultados similares al clasificador en solitario, siendo los de Stacking ligeramente mejores. Para el dataset Poker lo ensembles no mejoran los resultados del clasificador en solitario.

7. CONCLUSIONES

En ésta última sección se presentan las conclusiones sobre el trabajo realizado y los resultados obtenidos.

7.1 Conclusiones

En este Trabajo de Fin de Grado he realizado un estudio de la situación actual de la ciencia de datos, Big Data y *machine learning*. También se ha llevado a cabo un profundo estudio sobre el *framework* de Big Data de procesamiento de datos Apache Spark y en especial su módulo de *machine learning* MLlib.

Tras haber realizado toda la experimentación, los resultados obtenidos han sido satisfactorios ya que se ha demostrado que los algoritmos de ensembles *Stacking* y *Bagging* tienen la capacidad mejorar los resultados de las tasas de clasificación de los modelos en solitario sin tener que aumentar mucho el tiempo los tiempos de cómputo para datasets medianos-pequeños. También se ha demostrado que los tiempos de ejecución de los ensembles para un dataset grande se pueden reducir bastante si se paraleliza la ejecución de los modelos del ensemble.

Una vez finalizado el trabajo se ha subido el proyecto con el código fuente al repositorio de paquetes de Apache Spark *SparkPackages* (https://spark-packages.org) bajo el nombre *mllib-bagging-stacking*.

7.2 Conclusiones personales

Realizar éste trabajo me ha permitido introducirme en los campos de ciencia de datos, Big Data y *machine learning* que me llamaban bastante la atención y que hasta recientemente desconocía muchas cosas de ellos. Estoy satisfecho en este sentido porque he aprendido mucho sobre estos campos durante la realización de este trabajo.

El aprender Scala también me ha gustado porque con este lenguaje he aprendido cómo trabajar con programación funcional. El trabajar con Apache Spark me ha parecido interesante porque he visto cómo se trabaja con grandes cantidades de datos y lo que se puede hacer con ello mediante este *framework.*

Bibliografía

Apache Spark. Spark Release 2.3.3. (2019)
Recuperado de: https://spark.apache.org/docs/2.3.3/

UCI Machine Learning Repository, Center for Machine Learning and Intelligent Systems.
Recuperado de: http://archive.ics.uci.edu/ml/

[Dzeroski&Zenko2004] Dzeroski, S. & Zenko, B. Is combining classifiers with stacking better than selecting the best one? Machine Learning, 255–273. 2004.
Recuperado de:
https://link.springer.com/content/pdf/10.1023%2FB%3AMACH.0000015881.36452.6e.pdf

[Diettrich2000] Dietterich, TG. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning, 40(2), 139–158. 2000.
Recuperado de:
https://link.springer.com/content/pdf/10.1023%2FA%3A1007607513941.pdf

[Singh2018] Aishwarya Singh. A Comprehensive Guide to Ensemble Learning. 2018.
Recuperado de: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

[Gorman2016] Ben Gorman. Guide to Model Stacking (i.e. Meta Ensembling). 2016.
Recuperado de: https://www.gormanalysis.com/blog/guide-to-model-stacking-i-e-meta-ensembling/

[Brownlee2016] Jason Brownlee. Bagging and Random Forest Ensemble Algorithms for Machine Learning. 2016.
Recuperado de: https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/

[Xin2014] Reynold Xin. Apache Spark officially sets a new record in large-scale sorting. Databricks Engineering Blog. 2014.
Recuperado de: https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

[IBM2015] IBM. What is Big Data? IBM Analytics. 2015.
Recuperado de: https://www.ibm.com/analytics/us/en/big-data/

[Kovachev2019] Dilyan Kovachev. A Beginner's Guide to Apache Spark. 2019.
Recuperado de: https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92

[Phatak2015]Madhukara Phatak. History of Apache Spark : Journey from Academia to Industry. 2015.
Recuperado de: http://blog.madhukaraphatak.com/history-of-spark/

[Damji2016] Jules Damji. A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets. When to use them and why. 2016.
Recuperado de: https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

[IntelliPaat2017] IntelliPaat. Apache Spark Architecture. 2017.
Recuperado de: https://intellipaat.com/blog/tutorial/spark-tutorial/spark-architecture/

[Breiman1996] Breiman, Leo. "Bagging predictors." *Machine learning* 24.2: 123-140. 1996.

[Or2015] Andrew Or. Understanding your Apache Spark Application Through Visualization. 2015.
Recuperado de: https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html

[Meng2015] Xiangrui Meng. MLlib: Scalable Machine Learning on Spark. 2015.
Recuperado de: https://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf

[Alexander2013] Alvin Alexander. Scala Cookbook. O'Reilly Media, Inc. 2013.

[AnalyticsVidhya2018] What is Machine Learning? A Friendly Introduction for Aspiring Data Scientists and Managers. 2018.
Recuperado de: https://www.analyticsvidhya.com/machine-learning/

[Educba2019] What Is Data Mining? Advantages and Working of Data Mining. 2019
Recuperado de: https://www.educba.com/what-is-data-mining/

[Edureka2019] A Beginner's Guide to Data Science. 2019.

Recuperado de: https://www.edureka.co/bl10og/what-is-data-science/

Anexo I. Herramientas de desarrollo

En este anexo se muestran las herramientas que se han utilizado para el desarrollo del trabajo.

A.1 IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo integrado de la empresa JetBrains. Soporta varios lenguajes como Java, Scala, Go, Groovy etc.

Tiene dos versiones: la versión Community que es gratuita y la versión Ultimate que es de pago. Para el desarrollo de este trabajo se ha utilizado la versión Community.

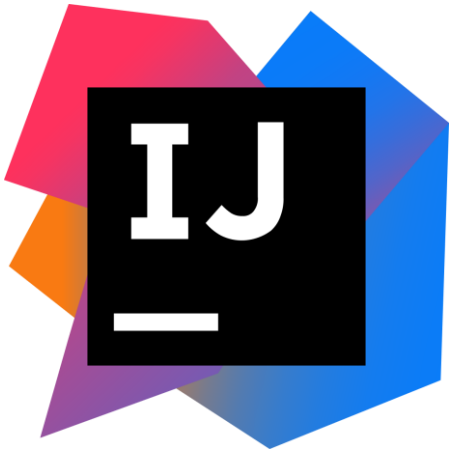

Figura A.1 Logo de IntelliJ IDEA

A.2 Git

Git es un software de control de versiones distribuido que permite llevar un registro de los cambios realizados sobre código o archivos.

En el desarrollo de este trabajo se ha utilizado para gestionar tanto el proyecto como la documentación.

Figura A.1 Logo de Git

A.3 GitLab

GitLab es una plataforma web de control de versiones y desarrollo de software colaborativo basada en Git. Esta plataforma ofrece servicios de gestión de repositorios, seguimiento de incidentes e integración continua.

Para el desarrollo de este trabajo se ha utilizado un repositorio privado que se ofrece de manera gratuita.

Figura A.3 Logo de GITLAB

A.4 GitHub

GitHub es otra una plataforma web de control de versiones y desarrollo de software colaborativo basada en Git. Esta plataforma ofrece servicios de gestión de repositorios, seguimiento de incidentes e integración continua.

Para el desarrollo de este trabajo se ha utilizado un repositorio público que se ofrece de manera gratuita ya que para la publicación del trabajo en https://spark-packages.org era necesario que el repositorio con el código estuviera específicamente alojado en esta plataforma.

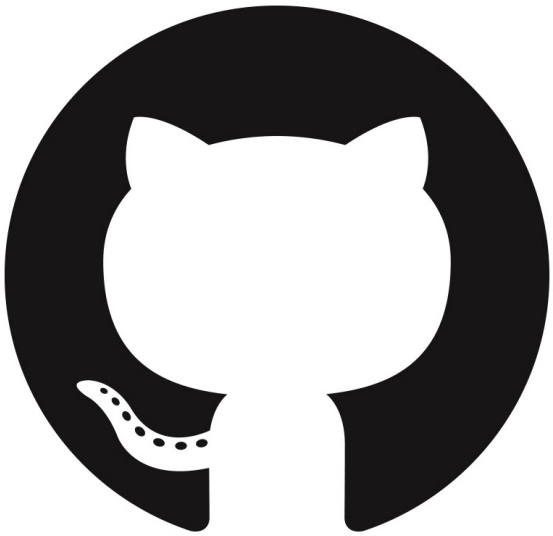

Figura A.4 Logo de GitHub

Anexo II. Manual de usuario

En este anexo se muestra cómo utilizar el proyecto subido al repositorio de paquetes de Apache Spark *SparkPackages* (https://spark-packages.org) bajo el nombre *mllib-bagging-stacking*.

B.1 Obtener el ejecutable del proyecto

Para la obtención de un jar a partir del código fuente es necesario IntelliJ IDEA y la versión compilada de Apache Spark 2.3.3.

El primer paso para obtener el ejecutable es abrir el proyecto en IntelliJ IDEA y abrir la opción *Build Artifacts…* dentro del menú *Build*.

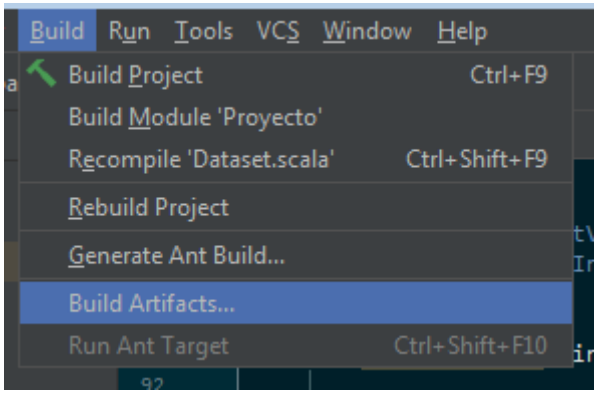

Figura B.1

Dentro del menú que aparece al pulsar esta opción se ha creado un nuevo artefacto como se muestra. Aquí es necesario incluir las bibliotecas de Apache Spark que se encuentran dentro de la carpeta jars de la versión compilada de Apache Spark 2.3.3 mencionada anteriormente. La configuración realizada se muestra en la Figura B.2.

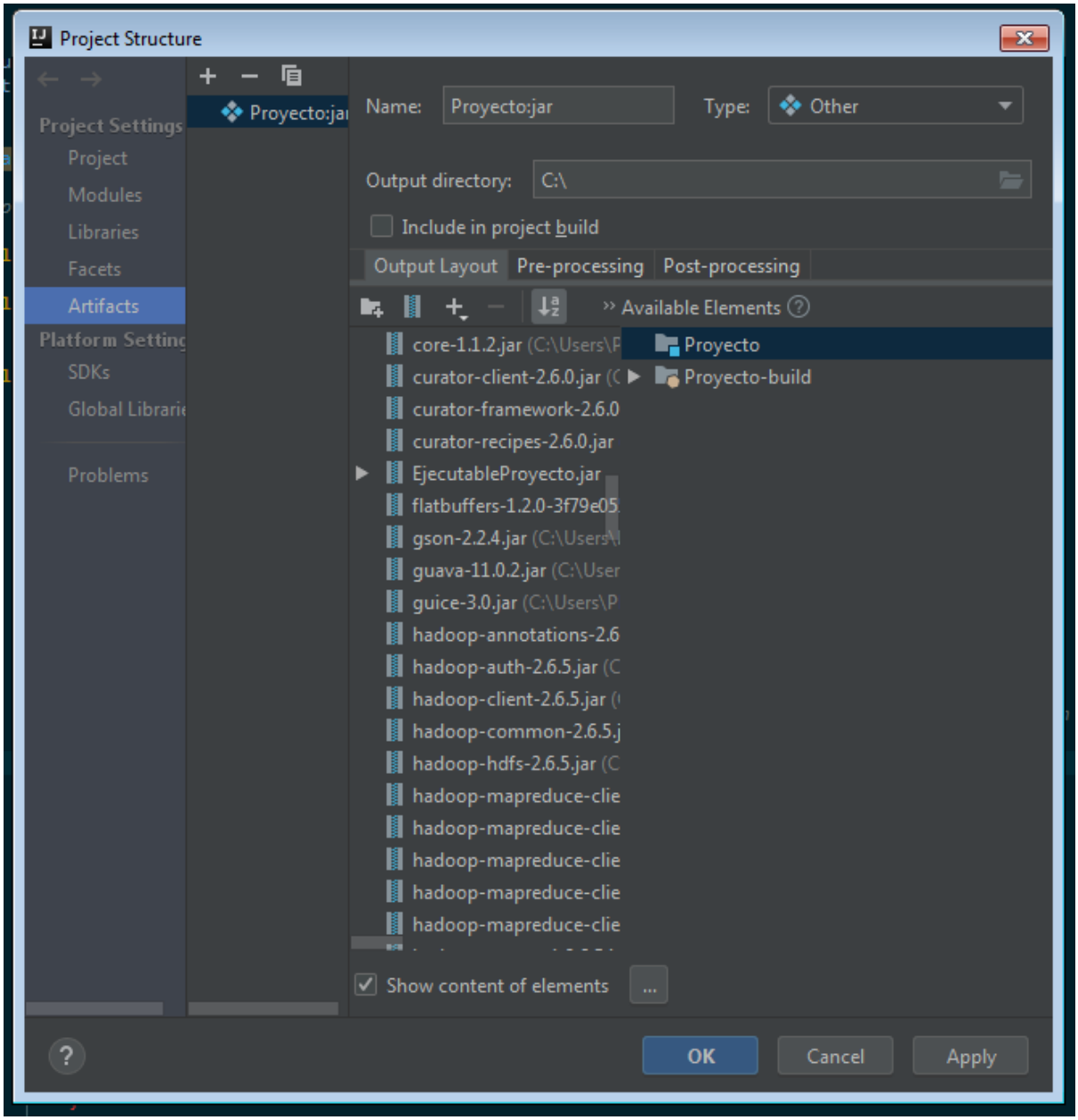

Figura B.2

Al compilar el proyecto, dentro de la carpeta de salida se generara un jar con el nombre EjecutableProyecto.jar, que es el ejecutable que contiene los ensembles.

B.2 Ejecución

Una vez obtenido el ejecutable EjecutableProyecto.jar, para hacer uso de él en un clúster con Apache Spark se utilizará el comando *spark-submit*.

A continuación se muestran la opciones de Spark para hacer uso del ejecutable en el clúster:

- --master: la URL del nodo master.
- --num-executors: número de ejecutores que se van a utilizar.
- --class: indica la clase principal de la aplicación, debe ser *prueba.MainBagging* para ejecutar *Bagging* y *prueba.MainStacking* para ejecutar *Stacking*.

El ejecutable tiene los siguientes argumentos de configuración:

- Primer argumento: dataset sobre el que se va a ejecutar el ensemble.
- Segundo argumento: fichero que se creará al finalizar la ejecución con los resultados obtenidos.
- Tercer argumento: número de particiones internas que se crearán al dataset.
- Cuarto argumento: modelos a utilizar en el ensemble. Cada modelo se marcará con –l0 y sus parámetros. Para Stacking será necesario incluir al final un modelo marcado con –l1 que será el modelo meta-learner.

Los parámetros que se pueden pasar a los modelos son:

Regresión Logística:

- Primer argumento: número de clases de salida del dataset.

Decision Tree:

- Primer argumento: número de clases de salida del dataset.
- Segundo argumento: número de contenedores utilizados al discretizar atributos continuos. El aumento de este parámetro permite que el algoritmo considere más candidatos divididos y tome decisiones más detalladas. Sin embargo, también aumenta el coste computacional. (Por defecto 32)
- Tercer argumento: profundidad máxima de un árbol. Los árboles más profundos permiten potencialmente una mayor precisión, pero también son más costosos de entrenar y tienen más probabilidades de *overfitting*.

Naive Bayes:

- Primer argumento: número que permite ajustar el *Additive smoothing*. (Por defecto 1.0).

A continuación se incluye un ejemplo para ejecutar Bagging sobre el clúster del CEATIC con el dataset Magic (con 2 clases) con 8 particiones internas y 5 Decision Tree como modelos a utilizar.

```
spark-submit
--master spark://bigdata:7077 \
--num-executors 8 \
--class prueba.MainBagging \
/home/simidat/pse00004/Proyecto_jar/EjecutableProyecto.jar \
hdfs://192.168.10.27:8020/user/simidat/magic-5-fold/magic-5-1tra.dat \
/home/simidat/pse00004/ResultadosMainBagging.txt \
8 \
-l0 DT 2 8 32 -l0 DT 2 8 32 -l0 DT 2 8 32 -l0 DT 2 8 32 -l0 DT 2 8 32
```