UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Paralelización de algoritmos de selección de instancias con la arquitectura Spark

Presentado por Alejandro González Rogel
en Universidad de Burgos — 28 de octubre de 2015
Tutor: Álvar Arnáiz González
Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Alejandro González Rogel, con DNI 71311632-V, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Paralelización de algoritmos de selección de instancias con la arquitectura Spark .

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de octubre de 2015

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de cuadros

Introducción

La minería de datos es un área que se ha mantenido en intensa y constante evolución desde su aparición formal en los años 80 y 90. Desde el comienzo, todos estos cambios han tenido por objetivo buscar una solución a los problemas que la minería de datos iba planteando. En lo que se refiere a la etapa actual, uno de los problemas más importantes a los que se está haciendo frente es la gran cantidad de los datos y el creciente número de atributos de los mismos [5], siendo estos los temas que se ha abordado a lo largo del proyecto.

El objetivo de este trabajo será la comparación de rendimiento entre las técnicas de minería en una ejecución lineal con respecto a la ejecución en paralelo, así como la implementación de diferentes algoritmos de reducción de instancias que, además de correr en paralelo, nos permitirán mejorar la calidad de los datos y reducir su cantidad.

Estructura de la memoria

La memoria ha mantenido la estructura definida en primera instancia por los profesores del tribunal del TFG:

- **Objetivos del proyecto:** Donde definiremos cuáles serán las metas que hemos intentado alcanzar con la realización del trabajo.
- **Conceptos teóricos:** Donde se darán a conocer todos aquellos conceptos que, sin estar incluidos dentro del conocimiento básico, son necesarios para la comprensión del proyecto.
- **Técnicas y herramientas:** Donde se explicarán las metodologías usadas para llevar a cabo el proyecto, así como cualquier herramienta que haya sido utilizada en la elaboración del mismo, incluyendo las causas de su elección en caso de considerarlo necesario.
- Se añadirán nuevas secciones.

Materiales entregados

Junto con la memoria se hará entrega de los siguientes archivos:

- **Anexo:** Documento adicional que contendrá el plan de proyecto,los requisitos diseño y los manuales de usuario y de programador.
- Se añadirán nuevos materiales.

Objetivos del proyecto

Este trabajo ha surgido como una prueba destinada a observar el rendimiento de algoritmos de *Big Data* en un entorno paralelo, así como a la implementación de diferentes algoritmos que permitan la selección de las mejores instancias durante la fase de pre procesamiento de la minería de datos.

2.1. Estudio del rendimiento de la minería de datos en un modelo de ejecución en paralelo

La minería de datos se trasladó a los entornos paralelos como una manera de poder tratar con grandes conjuntos de datos.

Como primera aproximación, utilizaremos dos herramientas diferentes: Weka (ver sección Weka) para la ejecución lineal y Spark (ver sección Apache Spark) para la ejecución en paralelo, para comparar el rendimiento de ambas modelos de ejecución frente a conjuntos de datos de diferentes proporciones.

2.2. Implementación de algoritmos de selección de instancias

Se programarán un conjunto de algoritmos que puedan aplicarse sobre grandes conjuntos de instancias con el fin de reducir dicho conjunto y mejorar el rendimiento de cualquier otro algoritmo que use posteriormente los datos. Para una definición más precisa de lo que es un algoritmo de selección de instancias ver la sección Algoritmos de selección de instancias

Se ha realizado la implementación de los siguientes algoritmos:

- Algoritmos implementados.

Conceptos teóricos

3.1. Minería de datos

Es el proceso mediante el cual podemos extraer conocimiento de un conjunto de datos que, sin ser tratados o analizados previamente, no nos proporcionan información útil [5].

Se trata de un término que a menudo puede generar confusión con el de KDD (Knowledge Discovery from Data), siendo en ocasiones tratado como un mero sinónimo de este término (que apareció antes que el de minería de datos) y en otras siendo descrito como un mero proceso dentro del descubrimiento de información, encargado de obtener conocimiento mediante la aplicación de algoritmos sobre datos recibidos [5].

KDD puede ser definido como una serie de pasos cuyo objetivo final es la extracción de información de un gran conjunto de datos [5]. Podemos agrupar dichos pasos en tres grandes fases: obtención y pre procesamiento de la información, aplicación de algoritmos de minería de datos y análisis y presentación de los resultados.

A lo largo de esta memoria trataremos a la minería de datos como sinónimo de KDD, esto es, el conjunto de procesos que comprenden desde el pre procesamiento de los datos hasta la obtención final de información útil.

3.2. Algoritmos de selección de instancias

El objetivo de estos algoritmos es solucionar dos problemas que afectan a la minería de datos: la cantidad cada vez mayor de datos, y su calidad. Podremos, por lo tanto, definirlos como una herramienta para extraer, de un conjunto de instancias, aquellas que conocemos, o sospechamos, son superfluas o perjudiciales [2].

Eliminando una porción del conjunto de instancias durante la fase de pre procesamiento de los datos conseguimos que el tiempo de ejecución algoritmos posteriores se reduzca, dado que hay menos instancias a examinar, mientras que es posible mejorar los resultados obtenidos al finalizar el proceso de minería [2].

Resilient Distributed Datasets (RDD)

Se trata de una de las características esenciales de Spark (es recomendable leer primero la sección sobre Apache Spark) y consiste en una colección de objetos, accesible en modo solo lectura, distribuida a lo largo de un conjunto de máquinas que pueden reconstruir una de sus particiones si esta llegase a perderse. [14]

Estas estructuras soportan dos tipos de operaciones:

- **Transformaciones:** Actúan sobre una estructura RDD produciendo como salida una nueva RDD resultado de una modificación de la anterior. Por defecto, todas las transformaciones sobre una RDD son perezosas (*lazy*), lo que quiere decir que no se ejecutarán hasta que una acción solicite un valor concreto que requiera de la transformación. [14]
- **Acciones:** Operaciones sobre las RDD que devuelven un resultado que depende del tipo de acción aplicada.

Otra de las grandes diferencias que caracterizan a las RDD de otro tipo de estructuras es la posibilidad de definir fácilmente el nivel de memoria en el que queremos alojar los datos, algo de lo que muchos frameworks anteriores carecían [13]. En un principio todas las RDD son efímeras, esto es, serán eliminadas de memoria si no se indica lo contrario, pero pueden mantenerse para obtener mejores resultados de rendimiento si los datos van a usarse repetidamente con relativa frecuencia. A esta acción de mantener en memoria una RDD se le llama cachear (caching).

Técnicas y herramientas

4.1. Técnicas

Scrum

Scrum es una metodología ágil de desarrollo iterativo e incremental para la gestión del desarrollo de un producto. [12]

Como parte de la metodología, el trabajo se ha dividido en springs, intervalos de tiempo de pocas semanas que ofrecen un producto al final de los mismos, que a su vez se han dividido en hitos y estos en tareas.

En lo que se refiere a su aplicación práctica dentro del proyecto, los springs han tenido una duración aproximada de dos semanas, periodo tras el cual había una reunión entre alumno y tutores para hablar sobre el avance y problemas ocurridos a lo largo del spring, así como para definir el avance del proyecto durante el próximo periodo de tiempo.

Para la gestión de los hitos y tareas nos hemos apoyado en el gestor de incidencias que la plataforma Bitbucket (ver Bitbucket) proporciona en el repositorio del proyecto. De esta manera, cada incidencia marcada con la etiqueta *task* corresponde con una tarea a realizar, mientras que todas ellas están agrupadas en hitos, llamados *milestones* en la plataforma.

4.2. Herramientas

Apache Spark

Apache Spark es un motor de interés general destinado al procesamiento distribuido de grandes conjuntos de datos. Está implementado en Scala, pero también proporciona APIs para otros lenguajes de programación (Java, Python y R) y otro tipo de herramientas para áreas como el aprendizaje automático (ver la sección Machine Learning Library (MLlib)) [9].

La idea nació como proyecto en 2010, en la Universidad de California, Berkeley, y su primera versión estable apareció el 30 de mayo de 2014. La motivación inicial era la de proporcionar un nuevo modelo de computación paralela que permitiera la ejecución eficiente de modelos que debían utilizar durante múltiples iteraciones grandes conjuntos de datos. Aproximaciones anteriores basadas en el modelo de MapReduce (como Hadoop), requerían cargar de nuevo todos los datos en memoria, haciendo la tarea demasiado costosa[14]. Como beneficio adicional, Spark ha demostrado que requiere de muchas menos líneas de código a la hora de programar algoritmos destinados al manejo de *Big Data*.

Actualmente Spark es un proyecto de código abierto cedido a Apache, siendo uno de los más activos en cuanto a contribuciones de la comunidad [6].

Para la realización del proyecto utilizaremos la versión de Spark 1.5.0.

Machine Learning Library (MLlib)

Se trata de una de las librerías incluidas en Spark. Contiene un conjunto de algoritmos de aprendizaje automático y algunas herramientas para ayudar en las labores de minería de datos, como tipos de datos o herramientas estadísticas.

Apache Hadoop

Es un framework escrito en Java destinado a el procesamiento distribuido de datos, tal y como hemos definido a Spark en la sección Apache Spark. Al igual que Apache Spark, se trata de un proyecto de código abierto [4].

Nació en 2006 de la mano de *Yahoo!*, quien más tarde lo cedería a Apache, como un proyecto que aplicase el modelo de programación de MapReduce que cuatro años atrás había definido Google [3]. Esta es la principal diferencia con Spark, quien ha dejado atrás el modelo anteriormente citado y ha seguido un nuevo camino gracias a las estructuras RDD.

Aunque Hadoop no ha sido utilizado directamente, se ha requerido de su instalación para el correcto funcionamiento de Spark. Aunque teóricamente ambos sistemas son independientes, existen en Spark algunas referencias a clases de Hadoop que pueden dar lugar a errores en el desarrollo de no encontrarse, razón por la que se ha decidido instalar.

La versión instalada es Hadoop 2.6.0.

Scala

Scala es un lenguaje de programación orientado a objetos y a la programación funcional y fuertemente tipado.

Es un lenguaje compilado, produciendo como salida ficheros .class que han de ser ejecutados en una máquina virtual de Java (JVM). Esto permite que librerías de Java puedan ser utilizadas directamente en Scala y viceversa. Por la misma razón, Scala posee la misma portabilidad que Java, pudiendo ejecutarse en cualquier sistema operativo siempre y cuando cuente con una máquina virtual de Java.

Los motivos de su elección como lenguaje de programación han sido mencionados anteriormente en la sección Elección del lenguaje de programación

Se ha usado Scala en su versión 2.11.7.

Java

Java es un lenguaje de programación orientado a objetos de propósito general diseñado para producir programas multiplataforma.

Necesitamos realizar la instalación de Java porque, aunque no trabajemos directamente sobre este lenguaje, si vamos a necesitar de su máquina virtual para poder ejecutar nuestros programas.

Hemos usado Java 8 u60 para la realización del proyecto.

JConsole

JConsole es una herramienta gráfica de monitorización para aplicaciones Java locales o remotas. Es una herramienta incluida dentro del Java Development Kit (JDK).

En el proyecto, se ha utilizado para evaluar y medir el rendimiento de aplicaciones en Java a nivel local.

La elección de utilizar esta herramienta frente a cualquier otra ha sido su facilidad de uso y la posibilidad de poder exportar a CSV las mediciones realizadas sobre el uso de memoria o CPU. Además, existe el hecho de que es una herramienta ya incluida en el JDK de Java.

JvisualVM

JvisualVM es una herramienta que proporciona información detallada sobre las aplicaciones Java que están corriendo en el sistema.

Su funcionalidad es practicamente similar a la de JConsole (ver **??**JConsole)), sin embargo, ofrece una mejor información sobre el estado de los hilos que componen una aplicación Java, siendo este el uso que se le ha dado a la aplicación.

Bitbucket

Es un repositorio de código que permite la creación, control y mantenimiento de proyectos, que podrán ser públicos o privados. Aunque Bitbucket ofrece la posibilidad de usarlo gratuitamente, también cuenta con otras posibilidades que solo se encontrarán disponibles en su versión de pago, como poseer un proyecto con un número ilimitado de colaboradores.

Puede trabajar con los sistemas de control de versiones Git y Mercurial.

Bitbucket fue propuesto como gestor del proyecto durante el primer spring del proyecto y, al no tener preferencia por ningún otro repositorio, se aceptó como herramienta a utilizar.

Git

Git es un sistema de control de versiones gratuito y de código abierto.

La elección de Git vino motivada por ser un sistema que ya había sido utilizado antes a lo largo de la carrera, por lo que no ha sido necesario aprender su funcionamiento.

Se ha utilizado la versión 2.6.2.

Eclipse

Eclipse es un entorno de desarrollo integrado (IDE de sus siglas en inglés) gratuito y de código abierto. Aunque su principal uso se basa en el desarrollo de aplicaciones en Java, también puede ser adaptado mediante el uso de plugins para ser utilizado en el desarrollo de otros lenguajes.

Como muchos otros entornos de desarrollo, posee como herramienta principal un editor de texto que en el caso de Eclipse cuenta con diferentes funcionalidades que pretenden apoyar al programador, poniendo como ejemplo el resaltado de texto, el autocompletado o la notificación y posibles soluciones de errores.

Un dato importante de este entorno de desarrollo es que está puramente basado en plugins, esto es, a excepción de un pequeño kernel, cualquier otra funcionalidad está incluida como un plugin, lo que le proporciona una gran facilidad para ser escalado o adaptado a las necesidades del usuario concreto.

Hemos trabajado sobre Eclipse 4.4.2.

ScalaIDE for Eclipse

Se trata de un plugin que puede añadirse al entorno de desarrollo Eclipse para poder desarrollar en Scala desde Eclipse. Este plugin consigue imitar la mayoría de los aspectos que Eclipse proporciona para Java para permitir un

desarrollo más cómodo, esto es, el autocompletado de código, resaltado de texto, definiciones e hipervínculos a clases, marcadores de errores y opción *debug*

La versión utilizada de este plugin es la 4.2.0.

Weka

Weka es un software desarrollado para llevar a cabo labores de minería de datos. Se trata de un proyecto de software libre, realizado en Java y desarrollado por la Universidad de Waikato, Nueva Zelanda.

Contiene, no solo algoritmos de aprendizaje automático para la minería de datos, sino también algoritmos de pre procesamiento de los datos o de visualización.

Al contrario que otras tecnologías que vamos a usar, esta librería no está pensada para la ejecución en paralelo, lo que la convierte en una buena herramienta para comparar el rendimiento que aplicaciones como Spark (véase **??**) pueden ofrecernos.

Se ha usado en su versión 3.6.13.

Zotero

Zotero es un gestor de referencias bibliográficas gratuito. Esta herramienta permite almacenar, de manera sencilla, referencias a un recurso concreto, permitiendo además crear una estructura de carpetas para clasificar las referencias o hasta compartir una biblioteca de referencias con otros usuarios registrados en el servicio.

La función de esta aplicación ha sido la de recompilar y organizar todos los enlaces que pudiesen ser de interés para la realización del trabajo y la memoria.

Para su uso se ha utilizado el plugin para el navegador Mozilla Firefox en su versión 4.0.

TeX Live

TeX Live es una distribución gratuita de LaTeX creada en 1996 y mantenida actualizada hasta la fecha. LaTeX, por su parte, es un sistema de creación documentos en los que se requiera una alta calidad tipográfica.

En el proyecto se ha utilizado LaTeX para la realización de la memoria, así como los documentos anexos.

TeX Live la distribución por defecto en muchos sistemas operativos Linux. Se ha utilizado su versión más reciente hasta la fecha, TeX Live 2015.

TexMaker

TexMaker es un editor multiplataforma pensado para el desarrollo de documentos escritos en LaTex. Al igual que otros muchos editores, esta plataforma presenta diferentes herramientas para hacer la creación de los documentos mucho más sencilla, tales como el autocompletado de etiquetas, la detección de errores ortográficos o el coloreado de texto.

Hemos usado la versión 4.4.1.

ProjectLibre

ProjectLibre es un programa de gestión de proyectos nacido como alternativa de código abierto a otros programas como Microsoft Project.

Está pensado para ayudar en el desarrollo de un plan de proyecto, en el seguimiento de dicho plan o en la gestión de recursos y cargas de trabajo.

En este proyecto ha sido utilizado para documentar el avance del mismo a lo largo del semestre.

Aspectos relevantes del desarrollo del proyecto

5.1. Elección del lenguaje de programación

Spark es un sistema que proporciona soporte a diferentes lenguajes de programación: Java, Scala, Python y, recientemente, R [9]. Eliminando este último como posible elección, por el desconocimiento del lenguaje y la poca documentación que hay sobre su uso con Spark, se ha realizado una comparativa entre las opciones restantes que podrían seleccionarse para llevar a cabo el proyecto. Los aspectos a tener en cuenta durante esta comparación han sido:

- **Experiencia previa:** Se tendrá en cuenta el contacto que se haya tenido con los lenguajes anteriormente.
- **Eficiencia de ejecución:** Valoraremos el rendimiento de cada lenguaje en función del tiempo que requieren para la ejecución de programas.
- **Facilidad de mantenimiento:** Las ventajas y facilidades del lenguaje en el caso de que se requiriese corregir o mantener un algoritmo o aplicación.
- **Adecuación:** Beneficios aportados por el lenguaje para su uso concreto en el desarrollo de algoritmos para Spark.
- **Documentación disponible:** Facilidad para encontrar información actualizada sobre el uso del lenguaje en Spark.

Nótese que la tabla 5.1 es una comparativa entre las características de los diferentes lenguajes para su uso en Spark, no una comparativa entre las

características propias de cada uno. Por lo tanto, aspectos que no tengan influencia en Spark o aquellos que sean iguales para todos los lenguajes, como, por ejemplo, la portabilidad, no serán incluidos en la comparativa. Así mismo, si dos lenguajes compartiesen una característica muy parecida o idéntica, esta será incluida una sola vez en la comparativa, fusionando las dos celdas que correspondan de la tabla 5.1.

Criterio	Java	Scala	Python
Experiencia previa	Se ha trabajado en Java múltiples veces durante el grado.	Es un lenguaje sobre el que nunca se ha trabajado.	Se han aprendido las nociones básicas durante la carrera.
Eficiencia de ejecución	Compila los ficheros generando archivos .class y los ejecuta sobre una máquina virtual de Java (JVM). Esto conlleva que la ejecución sea considerablemente más rápida que la del intérprete de Python.		Es un lenguaje interpretado, lo que afecta negativamente a su rendimiento. Sin embargo, su rendimiento en comparación con Scala mejora considerablemente si contamos con muchos procesadores.[7]
Facilidad de mantenimiento	Código más extenso, aunque la aparición Java 8, con elementos como las funciones lambda (ver 5.1), han mejorado este aspecto.	Menos líneas de código y una sintaxis más facilmente legible.	
Adecuación	Trabajar sobre Java nos obliga, a la hora de programar, a transformar estructuras y clases de Spark solo soportadas al trabajar en Scala o Python. Además, no cuenta con un intérprete interactivo.	Spark está pensado para trabajar sobre Scala o Python. De hecho, Spark ha sido creado en Scala, por lo que su conocimiento puede ayudar a lo largo del proyecto. Ambos lenguajes cuentan con un intérprete interactivo.	
Documentación disponible	Existe buena documentación en la página oficial de Spark, aunque es más escasa en otras fuentes. Además, muchas veces la documentación no trabaja sobre Java 8.	Existe una amplia documentación sobre el uso de ambos lenguajes en Spark.	

Cuadro 5.1: Comparativa entre características de Java, Scala y Python para trabajar sobre Spark.

La elección final del lenguaje a utilizar será Scala, argumentando lo siguiente sobre los puntos que hemos comparado:

- **Sobre la experiencia previa:** No se considera un problema aprender el lenguaje. Además, los archivos generados tras la compilación y, por consiguiente, la manera de ejecutarlos o monitorizar su rendimiento, es similar a Java, un lenguaje ya conocido.
- **Sobre la eficiencia de ejecución:** El rendimiento, en lo que a tiempo de ejecución se refiere, es muy similar a Java, por lo que se considera una ventaja frente Python.
- **Sobre la facilidad de mantenimiento:** En el momento de la elección del lenguaje no contamos con experiencia en el mantenimiento de un gran código de minería de datos, pero parece lógico que a menos cantidad de líneas que mantener, más fácil puede resultar la tarea.
- **Sobre la adecuación:** Tras probar Java y Scala con Spark se ha llegado a la conclusión de que el primero implica no solo más código, como era de esperar, sino también operaciones de conversión de estructuras que funcionan en Scala o Python pero son diferentes para Java. Además, frente a Java, Scala cuenta con un intérprete de comandos que nos permite hacer pruebas sin la necesidad de tener que generar y compilar el código cada vez que queramos probar algo.
- **Sobre la documentación disponible:** Scala cuenta, al igual que Python, con una extensa documentación actualizada. Al realizar pruebas con Java en Spark se ha notado un pequeño problema de falta de documentación, pero sobretodo, un problema para encontrar documentación actualizada para aspectos nuevos de Java 8 y que serán frecuentemente usados, concretamente las funciones lambda.

Funciones lambda en Java 8

En Spark, es habitual usar funciones como parámetros de muchas transformaciones y acciones que aplicamos sobre las RDD (ver Resilient Distributed Datasets (RDD)). Estas funciones, por lo general, son requeridas solamente para la operación concreta, por lo que se suelen definir directamente en el código.

Esto, antes de la llegada de Java 8, generaba un código semejante al siguiente:

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt").filter(
  new Function<String, Boolean>() {
    public Boolean call(String s) {
      return s.contains("error");
    }
});
```

Mientras que con Java 8 , el código se reduce a:

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt")
                          .filter(s -> s.contains("error"));
```

Dado que, como hemos dicho, este tipo de operaciones van a ser enormemente comunes en el desarrollo de algoritmos para Spark, el hecho de poder usar Java 8 puede reducir significativamente el número de lÍneas de código y, además, facilitar la comprensión del programa.

El uso continuo que se pueden dar a las funciones lambda en la programación con Spark ya se comprobó cuando se intentó programar con Java para Spark.

5.2. Comparativa entre las ejecuciones de clasificadores en Weka y Spark

Se ha realizado una comparativa entre el rendimiento que ofrecen Weka y Spark a fin de poder probar el rendimiento de Spark frente a alternativas anteriores que, como principal diferencia, no están pensadas para ser ejecutadas en paralelo.

Para realizar las mediciones utilizaremos conjuntos de datos de diferentes tamaños (detallados en la sección Conjuntos de datos), pero siempre un mismo algoritmo: el Naive Bayes. Naive Bayes es un algoritmo de clasificación probabilístico y relativamente simple que ya se encontraba implementado tanto en la librería de Weka como en la de Spark, razón por la cual ha sido elegido.

En un principio hemos supuesto que no habría grandes diferencias en cuanto a tiempo de ejecución o recursos entre ambas implementaciones.

La ejecución del algoritmo se analizará tanto en Weka como en Spark, siendo en este último ejecutado con diferente número de hilos: 1, 2 y 4.

El programa utilizado para medir el rendimiento ha sido JConsole (ver JConsole) en ambos casos, con el fin de utilizar la misma herramienta de monitorización en todos los experimientos. Para observar el comportamiento de los hilos hemos utilizado la herramienta JvisualVM. (ver **??**)

Criterios comparados

Los aspectos que hemos tenido en cuenta a la hora de recoger datos han sido:

- **Tiempo de ejecución:** El periodo analizado es aquel que incluye la lectura de datos, la preparación, entrenamiento y prueba de los diferentes clasificadores que genere la validación cruzada y el cálculo del resultado final. Esta manera de medir es algo que afecta esencialmente a Spark, que lanza un gran número de procesos al iniciarse y pierde algo de tiempo con respecto a Weka. Mediremos en segundos.
- **Memoria:** Media del espacio de memoria RAM que consume la ejecución del algoritmo. Se mide en megabytes.
- **Porcentaje de CPU:** Media del porcentaje de CPU utilizado con respecto a la potencia total de la CPU. Estas pruebas han sido realizadas sobre una máquina con capacidad para soportar 4 hilos al mismo tiempo, por lo que el uso completo de uno de los hilos supondría un porcentaje de carga de la CPU del 25 % con respecto al total, el uso exclusivo de dos hilos sería un 50 % y así sucesivamente.
- **Comportamiento de los hilos:** Como actúan los hilos del programa cuando se ejecuta el clasificador. Al ser un elemento que no puede medirse directamente, se recurrirá a hablar de él en la sección Conclusiones para sustentar otros argumentos, incluyendo fotografías que muestren el comportamiento en el caso de necesitarlas.

Entorno de las pruebas

Las mediciones se han llevado a cabo bajo las siguientes circunstancias:

- Las ejecuciones se realizaron desde la terminal, tanto en el caso de Weka como en el de Spark.

- En todas las ejecuciones hemos contado con memoria suficiente como para poder incluir en ella todo el conjunto de datos.
- Se necesita un tiempo para indicar a la herramienta de monitorización que proceso ha de medir. Por ello, con la intención de tener tiempo para asociar la herramienta de monitorización al proceso concreto, la ejecución hilo principal es suspendida durante un pequeño periodo de tiempo antes de iniciar siquiera la lectura de datos del fichero.
- Aunque no estamos interesados en la salida final del algoritmo, queremos simular que se realizan las fases de validación y test. No se ha utilizado un archivo diferente como conjunto de test, sino que hemos utilizado una validación cruzada de 10 iteraciones.
- El formato utilizado en los ficheros que contienen datos ha sido .arff para Weka y .csv para Spark. La razón por la que no se han utilizado ficheros .csv en Weka ha sido por la posibilidad de que esto produzca errores a la hora de leer el archivo [11].
- El formato de los datos de los conjuntos de datos también ha sufrido variaciones. Mientras que los archivos .arff contienen los datos tal y como los proporciona el data set original, los datos han tenido que ser normalizados para poder ser utilizados en Spark. La razón es que el algoritmo NaiveBayes implementado en Spark no admite atributos negativos y ha afectado a los conjuntos de datos "Human Activity Recognition", Çovertypez "HIGGS"(ver **??**Datasets))
- En lo que se refiere a Spark, se ha creado objeto en Scala que es capaz de leer el conjunto de datos, crear diferentes pliegues (*folds*) de dicho conjunto, entrenar y probar el clasificador y mostrar un resultado final. Weka ya proporciona herramientas de este tipo y por lo tanto no ha sido necesario generar ninguna otra clase.

Conjuntos de datos

Los conjuntos de datos, ordenados de menor a mayor según los recursos utilizados para correr el programa, han sido los siguientes:

Nombre del conjunto	Instancias	Atributos	Clases
Iris	150	4	3
Human Activity Recognition [10]	165632	17	5
Poker [8]	1.025.010	10	10
Covertype [8]	581.012	54	6
HIGGS [8] [1]	1.469.873 y 2.939.746[1]	28	2

Cuadro 5.2: Conjuntos de datos utilizados para la comparación entre Weka y Spark.

Indicar que, a la hora de seleccionar los conjuntos, se han elegido aquellos que compartan algunas características comunes:

- No existen campos de tipo texto. La única excepción la encontramos en los atributos de clase en los ficheros .arff, pero estos atributos serán tratados como nominales a la hora de la clasificación en Weka.
- No existen campos vacíos en ninguna de las instancias de los atributos.

Resultados

Los resultados obtenidos han sido agrupados según el conjunto de datos utilizado.

Las pruebas han sido ejecutadas dos veces bajo una misma configuración, de manera que podamos comprobar si el resultado obtenido se encuentra dentro de lo esperado o su buen/mal funcionamiento se debe a una circunstancia puntual.

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	0,1	1,97	2,13	2,15
Memoria(MB)	-	-	-	-
CPU(%)	-	-	-	-

Cuadro 5.3: Rendimiento sobre el conjunto de datos Iris.

[1] El conjunto original consta de 11.000.000 instancias, pero he reducido su tamaño original para acercarlo más al tamaño de los otros conjuntos.

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	12,37	16,13	11,75	11,71
Memoria(MB)	242,01	173,80	219,75	219,17
CPU(%)	25,5	36,02	54,03	59,43

Cuadro 5.4: Rendimiento sobre el conjunto de datos Human Activity Recognition.

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	73,90	47,44	31,84	30,77
Memoria(MB)	424,65	162,93	243,76	255,11
CPU(%)	30,05	29,41	51,22	55,68

Cuadro 5.5: Rendimiento sobre el conjunto de datos Poker.

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	183,95	101,91	73,23	53,78
Memoria(MB)	576,78	177,37	213,7	211,7
CPU(%)	28,17	28,26	43,10	71,94

Cuadro 5.6: Rendimiento sobre el conjunto de datos Covertype.

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	246,97	191,9	130,37	99,2
Memoria(MB)	832,88	872,56	864,92	921,49
CPU(%)	28,60	26,64	49,91	89,24

Cuadro 5.7: Rendimiento sobre el conjunto de datos HIGGS(1.469.873 instancias).

Criterio	Weka	Spark 1 hilo	Spark 2 hilos	Spark 4 hilos
Tiempo(s)	503,929	368,721	264,023	215,214
Memoria(MB)	1747,22	883,28	911,77	853,58
CPU(%)	29	26,37	50,74	91,58

Cuadro 5.8: Rendimiento sobre el conjunto de datos HIGGS(2.939.746 instancias).

Conclusiones

- Puede observarse claramente que Weka es considerablemente superior a Spark cuando utilizamos conjuntos de datos pequeños, como en la tabla 5.3 o incluso en la tabla 5.4, pero su rendimiento ve superado con notable diferencia cuando el conjunto de datos empieza a sobrepasar las 100.000 entradas de Human Activity Recognition(ver Conjuntos de datos).
- Salvo cuando tratamos con conjuntos de datos pequeños, vease por ejemplo la tabla 5.3, el rendimiento del algoritmos Naive Bayes de Weka es menor si lo comparamos con las ejecuciones sobre un solo hilo de Naive Bayes en Spark. Es posible que una de las causas sea la diferente implementación del algoritmo en Weka y en Spark.
- Como era de esperar, doblar el número de hilos no implica reducir a la mitad el tiempo de procesamiento, sino que genera un beneficio menor que, en algún momento y dependiendo del tamaño del conjunto de datos analizado, dejará de ser significativo aunque sigamos añadiendo hilos.
- Generalmente Spark necesita menos memoria que Weka para ejecutar el programa.
- Parece que el porcentaje de RAM requerido por Spark aumenta ligeramente cuantos más hilos tengamos en ejecución, algo que se aprecia bien en los conjuntos de datos pequeños.
- El porcentaje de uso de la CPU en Weka se situa siempre entorno a los valores 25-30 %. Esto es así porque la ejecución de todas las tareas es lineal, consumiendo únicamente un hilo de los 4 que posee la máquina en la que se están realizando las prueba.
- Vemos que el porcentaje de uso de la CPU en las diferentes pruebas con Spark suele corresponder al número de hilos con los que se lanza la aplicación: 25 % para un hilo, 50 % para dos y ,teóricamente, 90-100 % para 4. Sin embargo, y como podemos apreciar en las tablas 5.5 o 5.6 la ejecución con cuatro hilos no aprovecha al máximo las capacidades del

procesador cuando el conjunto de datos es pequeño. Explorando más de cerca el evento, vemos que, independientemente del número de hilos ejecutores que decimos a Spark que maneje, en estos conjuntos de datos únicamente se lanzan dos hilos como máximo. Atribuimos esto a un comportamiento propio de Spark, que evalua que no existe necesidad de manejar tantos hilos de ejecución.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] P. Baldi, P. Sdowski, and D. Whiteson. Searching for Exotic Particles in High-energy Physics with Deep Learning, 2014-7-2.

[2] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 2002. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.294.6862&rep=rep1&type=pdf.

[3] Databricks. Intro to Apache Spark, 2014. http://training.databricks.com/workshop/itas_workshop.pdf.

[4] The Apache Software Foundation. What Is Apache Hadoop?, 2015. https://hadoop.apache.org/.

[5] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[6] Jim Jagielski. Apache Software Foundation-Organization summary, 2015. https://www.openhub.net/orgs/apache.

[7] Peter Kerpedjiev. Python vs. Scala vs. Spark, 2015. http://emptypipes.org/2015/01/17/python-vs-scala-vs-spark/.

[8] M. Lichman. UCI Machine Learning Repository. *University of California, Irvine, School of Information and Computer Sciences*, 2013. http://archive.ics.uci.edu/ml.

[9] Apache Spark. Spark documentation, 2015. http://spark.apache.org/docs/latest/.

[10] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiú, and Hugo Fuks. Wearable Computing: Accelerometers' Data Classification of Body Postures and Move-

ments. http://groupware.les.inf.puc-rio.br/public/papers/2012.Ugulino.WearableComputing.HAR.Classifier.RIBBON.pdf.

[11] THe UNiversity of Waikato Weka. Can I use CSV files?, 2009. https://weka.wikispaces.com/Can+I+use+CSV+files%3F.

[12] The Free Encyclopedia Wikipedia. Scrum (software development), 2015. https://en.wikipedia.org/wiki/Scrum_%28software_development%29.

[13] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *University of California, Berkeley*, 2012. http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. *University of California, Berkeley*, 2010. http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf.