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 — 20 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, 20 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

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 la creciente dimensionalidad de los mismos, [4], siendo este el tema 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.

Probablemente se añadan pruebas con distribuciones de computación en la nube.

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.

Probablemente los algoritmos se usen también en las distribuciones de Spark que encontremos por la nube o se realicen mediciones de rendimiento.

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. [4]

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. [4]

KDD puede ser definido como una serie de pasos cuyo objetivo final es la extracción de información de un gran conjunto de datos [4]. 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.[1]

Eliminando un conjunto de instancias durante la fase de pre procesamiento de los datos conseguimos que el tiempo de 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. [1]

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, que se encuentra particionada a lo largo de un conjunto de máquinas que pueden reconstruir una de sus particiones si esta llegase a perderse. [10]

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 vagas (*lazy*), lo que quiere decir que no se ejecutarán hasta que una acción solicite un valor concreto que requiera de la transformación. [10]
- **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 [9]. Es un principio todas las RDD son efímeras, esto es, serán eliminadas de memoria si no se indica lo contrario, pero pueden mantenerse para obtener muchos mejores resultados de rendimiento si los datos van a usarse con relativa frecuencia.

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. [8]

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)) [7]

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[10]. 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. [5]

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. [3]

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. [2] 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

Elección del lenguaje de programación

Spark es un sistema que proporciona soporte a diferentes lenguajes de programación: Java, Scala, Python y, recientemente, R [7]. Eliminando este

último como posible elección, por el desconocimiento del lenguaje y la poca documentación que hay sobre su uso con Spark, vamos a realizar una comparativa entre las opciones restantes que podrían seleccionarse para llevar a cabo el proyecto. Los aspectos a tener en cuenta durante esta comparación serán:

- **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 siguiente (4.1) 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 4.1.

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.[6]
Facilidad de mantenimiento	Código más extenso, aunque la aparición Java 8, con elementos como las funciones lambda, han mejorado este aspecto.	Menos líneas de código tratarse de lenguajes de alto nivel.	
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 4.1: 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.

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.

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(vease **??**) pueden ofrecernos.

Se ha usado en su versión 3.6.13

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

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] 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.

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

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

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

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

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

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

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

[9] 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.

[10] 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.