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 — 7 de enero de 2016
Tutor: Álvar Arnaiz 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, 7 de enero de 2016

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

Resumen

La enorme cantidad de datos a los que ahora tenemos acceso han generado una serie de problemas cuando queremos extraer y estudiar la información contenida en dichos datos. Por esta razón, recientemente se ha empezado a desarrollar y mejorar nuevas técnicas que permitiesen resolver el problema, dos de las cuales serán analizadas y desarrolladas a lo largo de esta memoria: la computación paralela y los algoritmos de selección de instancias.

Usando Apache Spark, el objetivo principal de este proyecto será la comparación entre técnicas secuenciales y paralelas de minería de datos y la implementación y evaluación de dos algoritmos concretos de selección de instancias: *Locality sensitive hashing instance selection* y *Democratic instance selection*.

Descriptores

Minería de datos, big data, computación paralela, selección de instancias, Apache Spark, Locality sensitive hashing instance selection, Democratic instance selection.

Abstract

The huge amount of data which we have access to, has led to new problems when we try to analyze the information contained in that data. Because of that, new techniques have come out trying to solve the problem. Through this paper I will discuss about some of these new solutions: parallelization and instance selection algorithms.

Using Apache Spark engine, the main goal of this project will be the comparison between sequential and parallel data mining techniques and the implementation and evaluation of two concrete instance selection algorithms: Locality sensitive hashing instance selection and democratic instance selection.

Keywords

Data mining, big data, parallelization, instance selection, Apache Spark, Locality sensitive hashing instance selection, Democratic instance selection.

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, y debido a esta continua evolución, el área ha estado siempre sujeta a cambios que tenían 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 [10], que hacen imposible seguir aplicando aproximaciones anteriores por problemas de eficiencia.

A lo largo de esta memoria vamos a trabajar sobre algunos de los paradigmas que han surgido en el área para poder hacer frente a grandes volúmenes de datos: la paralelización de las tareas de minería de datos y la preselección de instancias para reducir el tamaño inicial del conjunto de datos y mejorar su calidad.

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 y herramientas usadas para llevar a cabo el proyecto, junto con una justificación de las causas de su elección.

- **Aspectos relevantes del desarrollo del proyecto:** Donde explicaremos todos aquellos apartados que consideremos de interés durante la evolución del proyecto.
- **Trabajos relacionados:**Donde se dejará constancia de cualquier otro trabajo que se hubiese realizado sobre el área tratada en este proyecto.
- **Conclusiones y líneas futuras de trabajo:** Donde se dejará constancia de todo lo aprendido o extraído del proyecto, así como de diferentes posibilidades para continuar trabajando.

Materiales entregados

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

- **Máquina virtual:** Se dejará a disposición del tribunal una imagen virtual de Ubuntu 14.04 que contendrá todos los materiales necesarios para probar el funcionamiento del proyecto.
- **Anexo:** Documento adicional que contendrá el plan de proyecto,los requisitos diseño y los manuales de usuario y de programador.

Objetivos del proyecto

En el escenario actual de la minería de datos (ver definición en 3.1) la creciente cantidad de datos que requieren ser analizados, así como el aumento de su complejidad, ha generado un problema a la hora de tratar esos datos con eficiencia y velocidad. El hecho de que este problema parezca ir en aumento ha dado lugar a una búsqueda de soluciones que proporcionen una alta capacidad de cálculo y una fácil escalabilidad [10].

Una de las propuestas ha sido la posibilidad de ejecutar las labores de minería de manera paralela. Por otro lado, analizando el problema desde un enfoque diferente aunque no incompatible, han adquirido gran popularidad los algoritmos encargados de reducir el número de instancias y atributos, de manera que podamos hacer más pequeño el conjunto a estudiar o incluso reducir su nivel de ruido. Este trabajo surge bajo la motivación de explorar estas nuevas soluciones.

Será un proyecto con dos objetivos principales: por un lado 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. Por otro, el análisis del rendimiento de implementaciones lineales frente a nuestras implementaciones paralelas.

Como objetivo adicional, surgido durante la realización del trabajo, tendremos en consideración la creación de un entorno visual y más intuitivo donde el usuario pueda ejecutar nuestro código de una manera más sencilla.

2.1. 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 su tamaño sin perjudicar

en exceso la información que transmiten. Para una definición más precisa de lo que es un algoritmo de selección de instancias ver la sección 3.2.

Se ha realizado la implementación de los siguientes algoritmos:

- *Localy sensitive hashing instance selection* (LSHIS) (ver 3.2)
- *Democratic instance selection* (DemoIS) (ver 8)

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

El principal objetivo de este estudio será comprobar los tiempos de ejecución según qué paradigma (secuencial o paralelo) y según que conjunto de datos.

Utilizaremos dos herramientas diferentes: Weka (ver definición en 4.2) para la ejecución lineal y Spark (ver definición 4.2) para la ejecución en paralelo.

2.3. Implementación de una interfaz gráfica de usuario

Con el objetivo de hacer más atractivo el uso de la aplicación y sus algoritmos, se ha incluido el desarrollo de una interfaz gráfica que permita plantear experimentos sin la necesidad de que el usuario tenga que acceder a su consola de comandos.

Dicha implementación no solo permitirá configurar de manera simple los algoritmos, sino que también será un entorno donde poder seleccionar algunas de las opciones de lanzamiento de Spark o incluso seleccionar los conjuntos de datos a utilizar.

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 [10].

Se trata de un término que podría 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 [10].

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 y presentación de información la información útil que contienen.

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 [5].

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 [5].

Localy sensitive hashing instance selection (LSHIS)

LSH, no confundir con el algoritmo de selección de instancias que definiremos a continuación, es un algoritmo que permite identificar y agrupar elementos muy semejantes. Su comportamiento se basa en un conjunto de funciones *hash* que tienen la característica fundamental la capacidad de asignar elementos similares a un mismo grupo (*bucket*) con una alta probabilidad [16].

El algoritmo LSHIS es un algoritmo de selección de instancias apoyado en el uso de LSH. La idea es aplicar, sobre las instancias iniciales, un conjunto de funciones hash que permitan agrupar en un mismo bucket aquellas instancias con un alto grado de similitud. Posteriormente, y realizando este proceso durante varias iteraciones si es preciso, de cada uno de esos buckets seleccionaremos una instancia de cada clase para formar el conjunto de instancias final.

La ventaja de este algoritmo frente a otras alternativas es que permite realizar la selección de instancias en un tiempo de complejidad lineal, en comparación con soluciones de complejidad cuadrática o logarítmica [16].

Algoritmo 1: LSH-IS – Algoritmo de selección de instancias mediante hashing. [16]

Input: Conjunto de instancias $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, conjunto $\mathcal{G}$ de familias de funciones hash
Output: Conjunto de instancias seleccionado $S \subset X$

```
S = ∅
foreach instancia x ∈ X do
    foreach familia de funciones g ∈ G do
        u ← cubeta asignada por la familia g a la instancia x
        if no existen otras instancias de la misma clase que x en u then
            Añadir x a S
            Añadir x a u
return S
```

Democratic Instance Selection

El algoritmo DemoIS es un algoritmo de selección de instancias que consiste en aplicar, durante un número variable de rondas, algoritmos de selección de instancias sobre subconjuntos disjuntos del gran conjunto inicial. En cada iteración, este algoritmo asignará unos "votos" a las instancias dependiendo de si han sido o no seleccionadas. Concluidas las rondas de votación, se realizará un cálculo con una función de *fitness* para seleccionar todas aquellas instancias cuyos votos no hayan superado un determinado límite [2].

Al igual que el algoritmo LSHIS mencionado anteriormente, la motivación fundamental es la de crear un algoritmo que requiera una carga computacional menor que las soluciones tradicionales. En este caso, eso se consigue mediante la división del conjunto de datos original en otros más pequeños y, por supuesto, el correcto análisis de los resultados anteriores.

Algoritmo 2: LSH-IS – Algoritmo de selección de instancias Democratic instance selection. [2]

Input: Conjunto de instancias $T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, tamaño de los subconjuntos s, número de rondas r
Output: Conjunto de instancias seleccionado $S \subset T$

```
for k = 1 to r do
    Dividimos las instancias en subjuntos disjuntos t_j de tamaño s tal que ⋃_i t_j = T
    for j = 1 to s do
        Aplicamos un algoritmo de selección de instancias a t_j
        Añadimos un voto a las instancias removidas de t_j
Calculamos la mejor función fitness en base a los votos
v = Votos que producen la mejor función fitness
S = T
Eliminamos de S todas las instancias cuyo número de votos sea >= v
return S
```

3.3. Computación paralela

La computación paralela es un tipo de computación en la que múltiples operaciones son llevadas a cabo simultáneamente [1]. Parte del principio de que algunos problemas pueden subdividirse en problemas independientes más pequeños que pueden resolverse al mismo tiempo.

Es un paradigma que desde el principio se usó para operaciones que requiriesen una gran carga computacional, pero que ha despertado mucho interés en los últimos años gracias a la fácil escalabilidad (ver 3.4) que puede ofrecer frente a otras alternativas, como el aumento de la frecuencia de los procesadores, que han alcanzado un punto donde resulta más difícil avanzar [20].

En lo que se refiere al uso de la memoria por parte de un sistema paralelo, existe la posibilidad de que la memoria sea compartida o distribuida dependiendo de si las unidades de procesamiento poseen un espacio de memoria común o cada unidad posee su propio espacio. En lo que se refiere a la tecnología que vamos a usar durante la realización del proyecto (ver 4.2) la memoria siempre será distribuida [24].

3.4. Escalabilidad

Escalabilidad es la capacidad de un sistema para adaptarse y soportar una carga de trabajo cada vez mayor [4].

En función de la manera de adaptarse a la nueva cantidad de trabajo, podemos diferenciar entre la escalabilidad horizontal, cuando añadimos más nodos a nuestro sistema, y vertical, cuando mejoramos las prestaciones de los elementos ya existentes [21]. Durante el desarrollo de la memoria siempre usaremos el término de escalabilidad para referirnos a la escalabilidad horizontal, pues estaremos hablando de sistemas distribuidos.

También es una característica que puede aplicarse a los algoritmos, para los cuales definimos escalabilidad como la capacidad de funcionar eficientemente cuando se aplican sobre situaciones que requieran una gran carga de trabajo. En lo referente al proyecto, esa situación será, por lo general, tratar con grandes conjuntos de datos.

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 [22].

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 springs, hitos y tareas nos hemos apoyado en el gestor de incidencias que la plataforma Bitbucket (ver 4.2) proporciona en el repositorio del proyecto. De esta manera, cada incidencia marcada con la etiqueta *task* corresponde con un hito a realizar, mientras que todos ellos están agrupados en springs, llamados *milestones* en la plataforma.

Hemos utilizado este método para realizar el seguimiento del proyecto porque, pese a no ser una herramienta especialmente dedicada a esta labor, evita el uso de nuevo software y puede ofrecer un buen resultado si existe atención por parte de los coordinadores del proyecto, en este caso los tutores [6].

4.2. Herramientas

Apache Spark

Apache Spark (http://spark.apache.org/) 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 4.2) [17].

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. 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* [24].

Actualmente Spark es un proyecto de código abierto cedido a Apache, siendo uno de los más activos en cuanto a contribuciones de la comunidad [12].

Para la realización del proyecto utilizaremos la versión de Spark 1.5.1.

Machine Learning Library (MLlib)

Se trata de una de las librerías incluidas en Spark. Contiene un conjunto de clases relacionadas con el campo del aprendizaje automático, como tipos de datos o funcionalidades estadísticas.

Resilient Distributed Datasets (RDD)

Es una de las características esenciales de Spark. Consiste en una colección de objetos, accesible en modo solo lectura y distribuida a lo largo de un conjunto de máquinas que pueden reconstruir una de sus particiones si esta llegase a perderse [24].

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 [24].

- **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 [23]. 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*).

Weka

Weka (http://www.cs.waikato.ac.nz/ml/index.html) 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 4.2) pueden ofrecernos.

Se ha usado en su versión 3.6.13 durante el principio del proyecto y en su versión 3.7.13 por problemas de compatibilidad con una de las librerías proporcionadas por el tutor.

Scala

Scala (http://www.scala-lang.org/) es un lenguaje de programación orientado a objetos y a la programación funcional. Es un lenguaje 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 5.1

Se ha usado Scala en su versión 2.11.7.

Java

Java (https://java.com/) 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. Además, hemos utilizado algunas de sus clases para realizar diferentes tareas de la aplicación, como partes de la interfaz gráfica.

Hemos usado Java 8 u60 para la realización del proyecto.

JConsole y JvisualVM

JConsole y JvisualVM son una serie de herramientas gráficas de monitorización para aplicaciones Java. Ambas están incluidas dentro del Java Development Kit (JDK).

En el proyecto, se han utilizado para evaluar y medir el rendimiento de aplicaciones en Java a nivel local. En el caso de JvisualVM, se ha utilizado específicamente para ver el estado de los hilos que componen la aplicación Java, algo que no permite hacer JConsole.

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

Bitbucket

Bitbucket (https://bitbucket.org/) 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 (https://git-scm.com/) 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 (https://eclipse.org/) 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.

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 [7] .

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 u opción *debug*.

La versión utilizada de este plugin es la 4.2.0.

Apache Maven

Apache Maven (https://maven.apache.org/) es una herramienta para la gestión y construcción de proyectos software en Java nacida con la intención de definir una manera estándar para la construcción de proyectos.

Se basa en el concepto de *Project Object Model* (*POM*), un archivo en formato XML que describe el proyecto a construir, la manera de construirlo y las dependencias con otros componentes.

El lenguaje utilizado para la elaboración del proyecto ha sido Scala (ver 5.1) y, como se ha mencionado, Maven fue pensado para trabajar sobre proyectos Java. Es por ello que necesitaremos de añadir un plugin adicional (Scala-Maven-Plugin) a nuestro fichero POM, aspecto que será tratado en el anexo entregado junto con la memoria.

Hemos utilizado esta herramienta para empaquetar los algoritmos preparados para Spark, además de para construir el propio Spark a partir del código fuente. Se ha seleccionado esta herramienta frente a otras opciones propias para Scala por conocerse con anterioridad su funcionamiento y por no requerir demasiado esfuerzo para ser utilizada en el lenguaje de programación que deseamos.

ScalaStyle

ScalaStyle (http://www.scalastyle.org/) es una plugin enfocado a la detección de medidas estáticas de calidad en el código de Scala.

Su funcionamiento se basa en la definición de una serie de reglas en un fichero .xml cuyo cumplimiento será revisado en todo el código del proyecto, indicando las líneas donde existe una irregularidad con respecto a dichas reglas.

Se ha incorporado este plugin tanto en Eclipse como en Apache Maven para detectar y corregir errores de calidad en el código.

Zotero

Zotero (https://www.zotero.org/) es un gestor de referencias bibliográficas gratuito. Es por esto que 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 en su versión 4.0 para el navegador Mozilla Firefox.

TeX Live

TeX Live (https://www.tug.org/texlive/) 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.

Se ha utilizado su versión más reciente hasta la fecha, TeX Live 2015.

TexMaker

TexMaker (http://www.xm1math.net/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.

Pencil Project

Pencil Project (http://pencil.evolus.vn/) es una herramienta de prototipado de interfaces gráficas.

Permite diseñar la apariencia de una aplicación arrastrando los componentes desde un menú de selección hasta una pantalla de dibujo, donde podremos modificar aspectos, como tamaño o texto, hasta conseguir el resultado que queremos. Es por ello que la función de esta aplicación es solo la de crear una imagen visual de nuestro prototipo, en ningún momento dicho prototipo tendrá funcionalidad ninguna.

He aplicado esta herramienta durante la realización de la interfaz gráfica del programa, para poder organizar los componentes en el espacio de una manera más sencilla.

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 [17]. 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.

Para elegir los aspectos a tener en cuenta durante esta comparación me he basado en el estándar ISO 9126 para la calidad del software [11]. Los puntos a analizar han sido finalmente los siguientes:

- **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 [13].
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 fácilmente 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 4.2). 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:

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

Código 5.1: Código de función lambda en Java 7 [14]

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

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

Código 5.2: Código de función lambda en Java 8 [14]

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 de rendimiento en la ejecución de clasificaciones entre Weka y Spark

Como primer aspecto a evaluar durante la realización del proyecto, se llevó a cabo una comparativa entre el rendimiento que ofrecen Weka y Spark.

La intención de esta prueba era doble: por un lado, suponía un primer acercamiento a la librería Spark y al modelo de funcionamiento en este tipo de entornos. Por otro, se pretendía probar el tiempo de ejecución de Spark frente

a alternativas anteriores que, como principal diferencia, no están pensadas para ser ejecutadas en paralelo.

Para realizar las mediciones utilizamos conjuntos de datos de diferentes tamaños (detallados en la sección **??**), 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.

Los resultados, así como una explicación más detallada del experimento, pueden encontrarse en la sección...

5.3. Implementación de algoritmos de selección de instancias

Marcada como el objetivo fundamental de este proyecto, la realización de dos algoritmos de selección de instancias ha ocupado gran parte de tiempo dentro del mismo. A continuación se expondrán algunos de los aspectos que más influencia han tenido durante esta fase de implementación y una pequeña explicación más concreta de la propia implementación de los algoritmos.

Indeterminismo, comunicación limitada y grandes conjuntos de datos

Por la propia naturaleza de la librería, la programación en Spark se ha diferenciado en muchos aspectos del tipo de programación secuencial que se ha aplicado hasta ahora. El hecho de que nuestros algoritmos estén pensados para ejecutarse en paralelo y distribuidos en una gran red de nodos nos proporciona numerosas ventajas, a las que ya nos referimos en las secciones y 2.3, pero también plantea problemas que es necesario tener muy en cuenta para la correcta ejecución del programa.

En primer lugar, una ejecución como la nuestra pierde la capacidad de predecir el orden en el que se ejecutarán las operaciones o el propio orden de las instancias cuando son distribuidas o intercambiadas por la red de trabajadores. Este problema ha obligado a modificar el planteamiento inicial de los algoritmos propuestos, pensados para ser ejecutados secuencialmente, y a adaptarlos al nuevo escenario.

Así mismo, y con la misma consecuencia, se nos plantea el problema de la comunicación entre nodos. Suponiendo una amplia red de nodos con un gran poder de computación en cada uno de ellos, es sencillo pensar que la comunicación entre todos ellos puede ser complicada si no queremos que esto afecte de manera muy negativa al rendimiento. Es por ello que Spark, pensado para la ejecución en este tipo de entornos, no proporciona demasiadas

posibilidades en este aspecto o, aquellas que ofrece, son bastante específicas o realmente costosas, como las operaciones de unión (*Join*). Todo esto ha sido tenido en cuenta durante la fase de implementación, afectando a las operaciones o incluso a la manera en la que almacenamos los conjuntos de datos, cuyas instancias han sido a menudo ligadas a diferentes valores que permiten controlar el flujo del programa.

Por último, en un intento de optimización por parte de Spark, nos encontramos dentro de la librería con un intento de evitar la aparición de operaciones costosas o, por lo menos, realizarlas de manera tal que no tengan un efecto tan perjudicial en el rendimiento. Esto tiene influencia en varios aspectos, siendo el más fácil de entender él de la distribución de las instancias entre los nodos. Algo tan aparentemente sencillo como la división de un conjunto de datos en particiones de igual tamaño es algo que no ha sido implementado en Spark por el altísimo coste y complejidad que supondría ejecutarlo eficientemente sobre un gran conjunto de datos. En cambio, se proporcionan estrategias basadas en tablas hash o el análisis de pequeños subconjuntos de prueba que permitan realizar la división en particiones aproximadamente iguales. Este tipo de limitaciones también han sido tenidas en cuenta de cara al desarrollo.

Por lo mencionado en este último punto, durante la realización del proyecto se han llegado a detectar funcionamientos anómalos de la librería Spark cuando opera con conjuntos de datos muy pequeños.

Normalización y pre procesamiento de los datos

Spark, y en particular MLlib, es una librería relativamente moderna que todavía no proporciona soporte para muchos tipos de operaciones. En lo que a este proyecto se refiere, se ha notado la falta de posibilidades para pre procesar los conjuntos de entrada que utilizamos para nuestros algoritmos.

Dado que esto podría suponer una carga de trabajo aún mayor y ya han tenido que implementarse clases adicionales para el funcionamiento de los algoritmos propuestos (ver 5.3), los conjuntos de datos utilizados necesitan cumplir dos requisitos para ser correctamente tratados:

- El conjunto de datos ha de contener solamente atributos numéricos, y esto incluye el atributo de clase.
- El conjunto de datos debe haber sido normalizado con anterioridad para una correcta clasificación.

Implementación de resursos necesarios

Como ya se ha comentado anteriormente, la librería MLlib de Spark con la que hemos estado trabajando aún se encuentra en una fase donde no contamos con una amplia colección de clases. Es por ello que han tenido que

implementarse algunos algoritmos adicionales a los propuestos como objetivo. Cabe destacar:

- **Algoritmo Condensed Nearest Neighbour (CNN):** Un algoritmo de selección de instancias simple y cuya ejecución es secuencial. Ha sido incluido de manera obligatoria para poder ejecutar correctamente nuestro algoritmo Democratic instance selector (ver 8). Además, es obligatorio que dicho algoritmo posea la capacidad de serialización, pues es necesario distribuirlo a cada uno de nuestros nodos trabajadores por exigencias del algoritmo que deseamos implementar.
- **Algoritmo K-Nearest Neighbours (KNN):** Un clasificador simple necesario tanto para la implementación de Democratic instance selector como para comparar la correcta salida de nuestros selectores de instancias. Está programado de manera secuencial, de manera que requiere que todos los datos sean recogidos en una sola máquina para poder aplicar el algoritmo. En un primer momento se creyó que podríamos contar con una implementación en Spark del KNN gracias al material presentado en la Conferencia de la Asociación Española para la Inteligencia Artificial 2015 (CAEPIA 2015), pero finalmente tuvo que implementarse una versión menos ambiciosa del clasificador.

Implementación concreta de LSHIS

Dadas las restricciones ya mencionadas en esta misma sección, la implementación del algoritmo LSHIS ha sufrido modificaciones con respecto a su propuesta original, cuyo pseudocódigo puede verse en 1.

En lo que a código se refiere, podemos ver una nueva versión del algoritmo en el pseudocódigo 3. El cambio fundamental puede apreciarse cuando existen dos o más construcciones-OR. Por culpa del indeterminismo que genera la ejecución en paralelo nos vemos obligados a ejecutar una serie de operaciones sobre conjuntos que no eran necesarias en la ejecución secuencial, donde se solucionaba el problema mediante el uso de un nuevo bucle *for*.

Cabe destacar que en estas operaciones realizadas cuando hay dos o más construcciones OR no usan en ningún momento el conjunto de instancias completo, sino que usan el conjunto solución generado por iteraciones anteriores, que se supone pequeño, y el conjunto solución generado por esta nueva iteración, que también debería tener un tamaño pequeño. Esta es una consideración muy importante en comparación con otras alternativas que surgieron, porque implica que la carga de trabajo va a ser mucho menor que si tuviésemos que operar con todo el conjunto de instancias inicial.

Como del pseudocódigo anterior no puede deducirse con claridad cómo están estructuradas las operaciones dentro del entorno paralelo, se incluyen los

Algoritmo 3: LSH-IS – Implementación paralela en Spark

Input: Conjunto de instancias $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, conjunto $\mathcal{G}$ de familias de funciones hash

Output: Conjunto de instancias seleccionado $S \subset X$

```
1  S = ∅
2  foreach componente OR do
3      g ← Nuevo conjunto de funciones hash
4      foreach instancia do
5          u, c ← cubeta asignada por las funciones hash g a la instancia x
               y clase de dicha instancia
6          Asociamos la instancia a su tupla u, c
7      instAgrupadas ← Conjunto de instancias agrupadas según su valor
       u, c
8      sel ← Conjunto con una instancia aleatoria por cada par llave u, c
9      if Es la primera iteración then
10         S = sel
11     else
12         foreach instancia en S do
13             u, c ← cubeta asignada por las funciones hash g a la
                   instancia x y clase de dicha instancia
14             Asociamos la instancia a su tupla u, c
15         sel' = sel − S ← Teniendo en cuenta la llave u, c asignada a
           cada instancia
16         S+ = sel'
17 return S
```

siguientes diagramas (ver 5.1 y 5.2) que permiten identificar como se realizan las operaciones dentro de un grupo de nodos. Compruébese que todas las operaciones se ejecutan en paralelo, en ningún momento ha sido necesario hacer ninguna operación secuencial. Los bloques de "Conjunto de datos inicial" y "Solución" simplemente han sido añadidos para aclarar el diagrama, pero tanto los datos iniciales como el resultado podrían perfectamente ser datos que ya se encontrasen distribuidos.

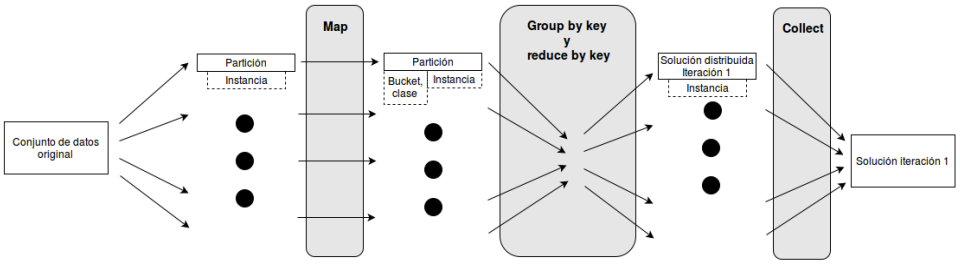

Figura 5.1: Diagrama de la ejecución del algoritmo LSHIS durante su primera iteración.

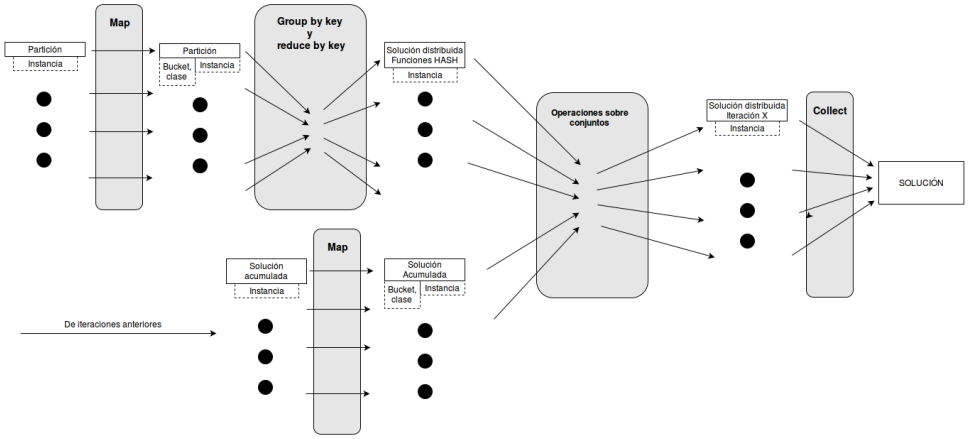

Figura 5.2: Diagrama de la ejecución del algoritmo LSHIS durante su segunda o mayor iteración.

Implementación concreta de DemoIS

La implementación de este algoritmo no ha requerido modificar el pseudocódigo del mismo (ver 8) pero sí existen varios detalles que conviene destacar sobre la actual implementación.

En primer lugar, como puede apreciarse en la figura 5.3, no todo el algoritmo se ejecuta en paralelo. A la hora de calcular el *fitness* óptimo lo hacemos de manera secuencial. Esto es así porque no contamos con una implementación paralela del algoritmo KNN, necesario para este proceso (ver 5.3). Aún con esto, durante esa etapa del proceso solo seleccionamos un pequeño grupo de instancias en comparación con el conjunto inicial, por lo que operar con tal número de datos no supone un problema de rendimiento tan grande.

Existen otras consideraciones de cara a la implementación, como el uso de un particionador aleatorio que no genera particiones del mismo tamaño o la creación de una clase individual que evite la serialización completa del algoritmo al realizar la primera operación *map*. Sin embargo, estos temas serán tratados con más detenimiento en el anexo incluido junto con la memoria.

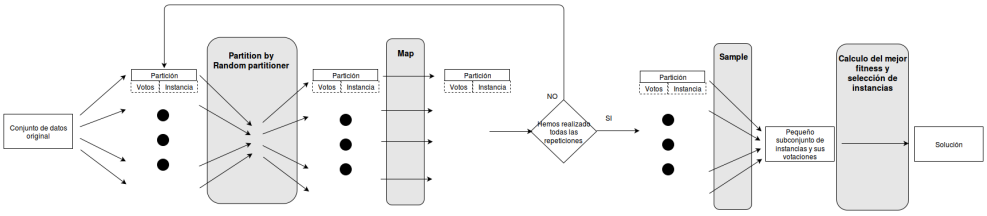

Figura 5.3: Diagrama de la ejecución del algoritmo DemoIS.

5.4. Comparativa entre las implementaciones de Weka y Spark

Como otro de los objetivos principales del proyecto, se procedió a realizar una comparación entre nuestras implementaciones de los algoritmos de selección de instancias con las ya existentes para su uso en Weka.

De nuevo, la finalidad de esta comparación tenía varios objetivos: evaluar el rendimiento entre ambas aproximaciones, comprobar el correcto funcionamiento de los algoritmos y evaluar como los cambios de implementación podían haber afectado a su funcionalidad.

Existe una sección más detallada donde se profundizará sobre todos los aspectos relacionados con la comparativa en la sección...

5.5. Implementación de un entorno gráfico

En la etapa final, se vio como el proyecto había alcanzado una complejidad considerable en cuando a las opciones de lanzamiento se refiere. Este hecho hacía de la ejecución por línea de comandos un método de lanzamiento mucho más tedioso de lo que a un usuario normal podría resultarle ya de principio. Es por esta razón que, con la intención de facilitar el uso de la biblioteca, que se decidió la implementación de una pequeña interfaz gráfica que hiciese más intuitívo el uso del proyecto.

Así pues, se ha realizado una interfaz gráfica que permita introducir, mediante campos de texto, todas las opciones necesarias para la ejecución de los algoritmos, así como la posibilidad de seleccionar los diferentes conjuntos de

datos o selectores de instancias. Además, posibilita la acción de crear baterías de ejecuciones al dar la opción de indicar más de una configuración de Spark, conjunto de datos y/o filtro a la vez.

Esta interfaz ofrece dos modos de ejecución:

- Permite ejecutar los algoritmos directamente desde la interfaz gráfica. Es una opción no recomendada si lo que se busca es eficiencia en los tiempos de ejecución pero es una manera sencilla de realizar pruebas sobre conjuntos de datos pequeños
- Permite la compresión en un archivo de extensión .zip de todos los conjuntos de datos necesarios para llevar a cabo las ejecuciones y de un archivo .sh generado dinámicamente que permita la ejecución de todas las ejecuciones seleccionadas. Este tipo de operación facilita que el usuario pueda definir todas las operaciones desde su máquina y, posteriormente, pueda trasladarlas de manera sencilla a un servidor más potente donde serán ejecutadas.

Trabajos relacionados

6.1. Weka

Aunque ya ha sido mencionado con anterioridad en esta memoria (ver 4.2), parece obligatoria una mención a dicho programa en este apartado.

Cabe destacar que, aunque tanto Weka como este proyecto comparten un objetivo común como es el de presentar una biblioteca con algoritmos de minería de datos y hacerlo de manera que sea fácil su utilización, el ámbito de aplicación de ambos programas es completamente distinto. Mientras que Weka parece pensado enfocado a pequeños/medianos conjuntos de datos y una ejecución secuencial de sus algoritmos, este proyecto utiliza Spark para hacer factible la idea de aplicar algoritmos demasiado costosos como para ser ejecutados con grandes conjuntos de datos en una sola máquina.

6.2. Documentos científicos de LSHIS y DemoIS

A lo largo del proyecto, y en especial todo lo relacionado con la implementación y prueba de los algoritmos, se ha contado con los documentos científicos que definían la implementación y uso de LSHIS y DemoIS. Dichos documentos, ya mencionados anteriormente en la memoria, han sido "LSH-IS: Un nuevo algoritmo de selección de instancias de complejidad lineal para grandes conjuntos de datos"[16] y "Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts"[2].

Ambos documentos, además de contener información sobre los algoritmos implementados, también contienen comparaciones de rendimiento con otras alternativas de selección de instancias. En lo que se refiere a este proyecto, las pruebas realizadas han tenido como objetivo la comparativa entre tiempos de ejecución de las implementaciones secuencial y paralela, por lo que podemos defender que se han evaluado aspectos diferentes durante la etapa de experimentación.

Conclusiones y Líneas de trabajo futuras

Bibliografía

[1] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989.

[2] Álvar Arnaiz-González, José F. Diez Pastor, César García-Osorio, and Juan J. Rodrígez. Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Elsevier*, Volume 174:410–441, 2010.

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

[4] André B. Bondi. *Characteristics of Scalability and Their Impact on Performance*. ACM, 2000. http://doi.acm.org/10.1145/350391.350432.

[5] 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.

[6] Cedric Dugas. Agile workflow with GitHub issues, 2014. http://www.position-absolute.com/articles/agile-workflow-with-github-issues/.

[7] Scala IDE for Eclipse. ScalaIDE for eclipse. http://scala-ide.org.

[8] Salvador García, Joaquín Derrac, and Francisco Herrera. A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification, 2011. http://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/1365_2012-Triguero-IEEETSMCC.pdf.

[9] Salvador García, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. Prototype Selection for Nearest Neighbor Classification: Survey of

Methods, 2012. http://sci2s.ugr.es/sites/default/files/files/TematicWebSites/pr/T-4-2010-PSMethods.pdf.

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

[11] ISO. Software engineering – product quality. ISO 9126-1:2003, International Organization for Standardization, Geneva, Switzerland, 2008.

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

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

[14] Justin Kestelyn. Making Apache Spark Easier to Use in Java with Java 8, 2014. http://blog.cloudera.com/blog/2014/04/making-apache-spark-easier-to-use-in-java-with-java-8/.

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

[16] José Miguel Puerta, José A. Gámez, Bernabé Dorronsoro, Edurne Barrenechea, Alicia Troncoso, Bruno Baruque, and Mikel Galar, editors. *LSH-IS: Un nuevo algoritmo de selección de instancias de complejidad lineal para grandes conjuntos de datos*, 2015.

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

[18] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiú, and Hugo Fuks. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. http://groupware.les.inf.puc-rio.br/public/papers/2012.Ugulino.WearableComputing.HAR.Classifier.RIBBON.pdf.

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

[20] Wikipedia. Parallel computing — Wikipedia, The Free Encyclopedia, 2015. https://en.wikipedia.org/w/index.php?title=Parallel_computing&oldid=688312615.

[21] Wikipedia. Scalability — wikipedia, the free encyclopedia, 2015. https://en.wikipedia.org/w/index.php?title=Scalability&oldid=679390144.

[22] The Free Encyclopedia Wikipedia. Scrum (software development), 2015. https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=688230392.

[23] 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.

[24] 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.