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
# Documentación Técnica

Presentado por Alejandro González Rogel
en Universidad de Burgos — 1 de febrero de 2016
Tutor: Álvar Arnaiz González
Carlos López Nozal

# Índice general

# Índice de figuras

*Apéndice A*

# Plan de Proyecto

## A.1. Introducción

A lo largo de este apéndice se va a presentar tanto la evolución temporal que ha seguido el proyecto durante su realización como aquellos aspectos de viabilidad que puedan afectar al trabajo si se desease continuar con él en un futuro.

## A.2. Planificación temporal

En esta sección vamos a presentar un resumen de lo que ha sido la realización del proyecto.

El desarrollo del proyecto se ha llevado a cabo mediante el uso de metodologías ágiles, realizando reuniones periódicas con los tutores para debatir sobre el desarrollo del trabajo y definir nuevos objetivos. Estas reuniones se han realizado, salvo excepciones, de manera bisemanal, y suponían el fin de un periodo de trabajo (*sprint*) y el comienzo del siguiente. Durante estas reuniones, además, se presentaba el trabajo realizado durante la última iteración.

Con el fin de no expandir más de lo necesario este anexo, solo se presentará una breve descripción del trabajo realizado en sprint del proyecto. Información más amplia sobre cada uno de los hitos y/o tareas puede encontrarse en la sección *Issues* del repositorio del proyecto, donde los hitos formarán un *issue* y las tareas, así como conversaciones entre los miembros del proyecto, pueden verse recogidas dentro de dicho *issue*.

### Sprint 1 [29/09/15-13/10/15]

Partiendo de un estado de desconocimiento de la tecnología a utilizar durante el desarrollo, se ha invertido gran parte del tiempo en aprender el fun-

cionamiento de Spark y la terminología del área de trabajo.

Igualmente, se realiza el despliegue de todos los materiales necesarios para el desarrollo. Algunos de ellos no fueron necesarios para iteraciones siguientes, como Python o Hadoop, y han sido eliminados más adelante. Además, el despliegue se realizó por duplicado, al encontrar problemas con la ejecución de Spark en el sistema operativo Windows.

Finalmente, se presentan dos pequeños programas (uno en Java y otro en Scala) para comprobar el funcionamiento de Spark y poder decidir el lenguaje a usar durante la etapa de desarrollo.

### Sprint 1 [13/10/15-29/10/15]

Se selecciona Scala como lenguaje de programación y comenzamos a aprender las bases de dicho programa.

Como objetivo principal de este sprint se plantea realizar una comparativa entre el tiempo de ejecución entre Weka y Spark con el algoritmo Naive Bayes. Todo ello genera una serie de hitos, definidos en las líneas sucesivas.

Se busca una serie de conjuntos de datos con diferentes características de tamaño/número de atributos, que además cumplan una serie de condiciones, como la ausencia de valores nulos. A la mayoría de los conjuntos seleccionados, pese a todo, hubo que pre procesarlos para que pudiesen funcionar sin problemas con Spark.

Se genera una pequeña clase lanzadora en Spark para poder lanzar y medir nuestros experimentos. Del mismo modo se realizan pequeñas modificaciones en el código de Weka para poder realizar correctamente mediciones.

Además, el comienzo de la documentación forzó al aprendizaje de LaTeX.

Finalmente, se terminó el sprint con la entrega del algoritmo en Scala utilizado para las mediciones y un informe con la explicación, resultados y conclusiones de la comparación Weka-Spark.

### Sprint 1 [29/10/15-19/11/15]

A partir de este momento comenzamos a centrarnos en otro de los objetivos principales del proyecto: la implementación de algoritmos de selección de instancias.

Se implementa una primera versión del LSHIS y se generan diferentes métodos para realizar labores como la lectura de datos. Es la primera implementación con cierta complicación que realizamos tanto en Scala como para Spark, por lo que este hito ya se plantea con una carga de trabajo importante.

Al final del sprint contamos con una implementación de una primera versión funcional del algoritmo LSHIS.

### Sprint 1 [19/11/15-10/12/15]

Se reestructura todo el código de la iteración anterior, creando nuevos paquetes, clases y relaciones de herencia. Además, se empieza a utilizar herramientas para el control estático de la calidad del código, lo que fuerza algunos cambios para adaptarnos a los estándares fijados.

El propio algoritmo LSHIS sufre una mejora, permitiendo ahora ejecutarlo con uno o varios componentes OR.

Se comienza la implementación del algoritmo DemoIS, pero queda suspendida a la espera de contar con una implementación paralela del algoritmo $k$-Nearest Neighbors ($k$NN).

Igualmente frenada por esta espera del algoritmo KNN, se plantea una comparación entre nuestra implementación LSHIS y la ya realizada en Weka, aunque nunca llega a producirse dentro de este sprint.

Se crea una máquina virtual con todos los materiales necesarios para el despliegue y distribución sencilla de nuestra aplicación.

Al finalizar este sprint contamos con una versión mejorada del algoritmo LSHIS, una mejor estructuración de nuestro código y un entorno para la fácil distribución de nuestro trabajo.

### Sprint 1 [10/12/15-22/12/15]

Se implementa de manera completa el algoritmo DemoIS. Esto obliga a la creación de algunos otros componentes como el algoritmo KNN, cuya implementación paralela no pudimos conseguir.

Se realizan pruebas sobre nuestro algoritmo LSHIS. Dichas pruebas generan un número considerable de problemas y, finalmente, cuando conseguimos realizar las mediciones, éstas arrojan malos resultados.

La máquina virtual sufre modificaciones para facilitar la ejecución del proyecto dentro de la misma.

Tras el sprint, podemos mostrar un nuevo algoritmo de selección de instancias y una máquina virtual mejor preparada para permitir la ejecución del proyecto.

### Sprint 1 [22/12/15-08/01/16]

Se realiza una primera implementación de la interfaz gráfica, que acaba por afectar a prácticamente la totalidad de la biblioteca, aunque nunca de manera sustancial.

Las pruebas sobre nuestros algoritmos siguen siendo problemáticas y continuamos trabajando sobre ello. Durante este periodo se encuentran una serie de

errores que afectaban a la manera en la que medíamos el tiempo de ejecución, así como unos errores en los algoritmos de Weka con los que intentábamos comparar nuestros resultados.

Finalmente, comenzamos a estudiar como lanzar nuestra ejecución en algún servicio de computación en la nube, pero no llegamos a finalizar la tarea al considerar más prioritario acabar con los problemas descritos anteriormente.

Por lo tanto, al terminar esta etapa contamos con una nueva manera de ejecutar nuestra aplicación (mediante la interfaz gráfica) y una serie de errores importantes corregidos, tanto en nuestra aplicación como en la librería de Weka facilitada.

### Sprint 1 [08/01/16-25/01/16]

Se realiza una nueva iteración sobre nuestra implementación de los algoritmos, esta vez centrada en mejorar el tiempo de ejecución de los mismos, que continua siendo muy elevado con respecto a Weka. Es algo que se resolverá en este periodo.

Habiendo solventado todos los problemas, comenzamos, de nuevo, a realizar pruebas sobre el rendimiento de nuestro trabajo.

Igualmente, iniciamos otra tarea que había quedado suspendida, el despliegue del proyecto en un clúster real, lanzándolo finalmente en dos servicios diferentes: Google Beta Dataproc [12] y un servicio clúster con el que contaba la Universidad de Burgos.

Durante este periodo también se trabaja en una gran refactorización del código, cuya calidad había disminuido considerablemente con todos los cambios introducidos en el sprint anterior.

Al finalizar esta iteración contamos con unos algoritmos que han demostrado funcionar correctamente y los resultados de una serie de comparativas que hemos realizado sobre nuestras implementaciones.

### Sprint 1 [25/01/16-05/02/16]

Entre este periodo y el anterior, se mantienen una serie de ejecuciones de nuestros algoritmos en el servicio clúster con el que cuenta la universidad, pero acaban cancelándose por problemas con los nodos.

Se realizan pequeñas modificaciones en el código en vista a mejorar su calidad o corregir algún comportamiento indeseado.

Para terminar, se prepara la versión final de todos los materiales necesarios para la presentación del proyecto al finalizar este último sprint.

## A.3. Estudio de viabilidad

El trabajo presentado a lo largo de este proyecto podría enfocarse de dos maneras distintas en esta sección: por un lado, podemos hablar de la viabilidad de los algoritmos implementados. Por otro, de las bases que este proyecto podría sentar para construir una librería con algoritmos de selección de instancias que corran en paralelo.

Sobre el primer punto, la viabilidad de los algoritmos, podemos basarnos en su rendimiento (estudiado en C.5) para concluir que, aunque una de nuestras implementaciones (LSHIS) no aporta una gran mejoría en cuanto a tiempo de ejecución se refiere, la otra (DemoIS) puede llegar fácilmente a mejorar los tiempos de respuesta de su predecesora. Además, ambos algoritmos están pensados para su ejecución sobre grandes conjuntos de datos, algo que no pueden soportar las versiones anteriores de estos algoritmos y que nos proporciona una ventaja fundamental en un ambiente donde cada vez existen un mayor número de datos que analizar.

Sobre el proyecto como una base para formar una librería, supondría, no solo trabajar en un área en continua expansión, sino comenzar en un momento en el que no existen otras grandes librerías que trabajen con Spark, lo que nos proporciona una gran flexibilidad para orientar el proyecto a donde se estime conveniente.

En lo que se refiere a aspectos de viabilidad es interesante referirse también al posible futuro de la tecnología en la que se basa el proyecto: Spark.

Aunque Apache Spark es una librería cuya versión inicial se presentó a mediados de 2014, a gozado de buena acogida desde entonces, siendo actualmente uno de los proyectos mejor valorados de la fundación Apache Software Foundation y uno de los más activos [13]. Igualmente, podemos observar en la gráfica A.1 como el interés por esta tecnología ha crecido en un corto periodo de tiempo para equipararse en interés a muchas de las soluciones ya existentes en el mercado. Nos encontramos, por lo tanto, operando con una tecnología que promete seguir desarrollándose gracias a la aceptación con la que ha sido recibida.

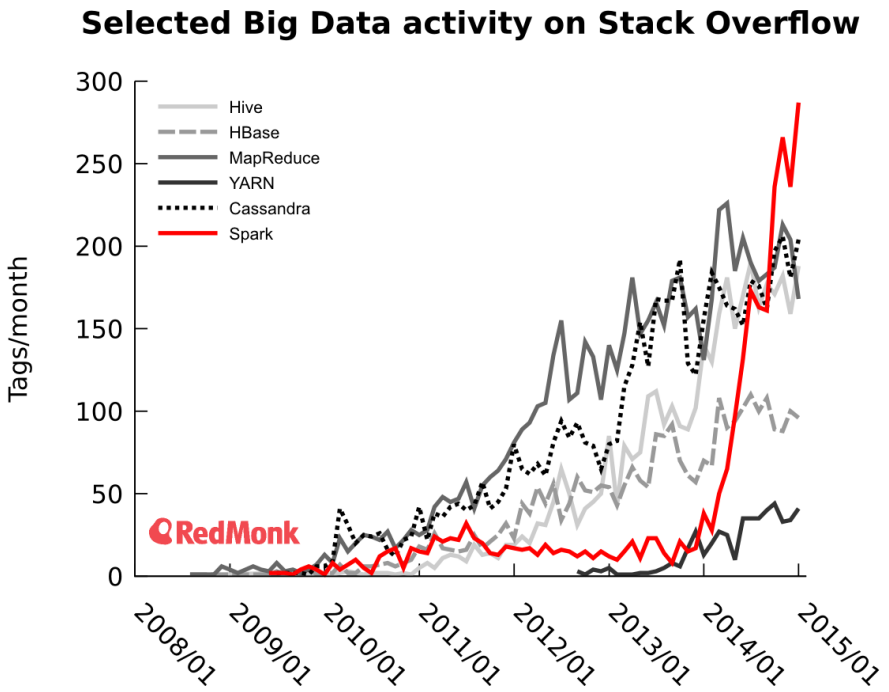


Figura A.1: Evolución del interés por Apache Spark en la plataforma Stack Overflow[7]

### Viabilidad económica

Aunque este proyecto nunca ha sido enfocado a un aspecto de comercialización inmediata, podemos ver un escenario favorable en el sentido económico.

La minería de datos es un ámbito del que se espera rápida y fuerte expansión. Podemos observar en la gráfica A.2 como la intención de muchas de las empresas es la de continuar apostando por la inversión en minería de datos, que ha llegado a convertirse en una prioridad para una gran cantidad de empresas de sectores muy diversos [8].

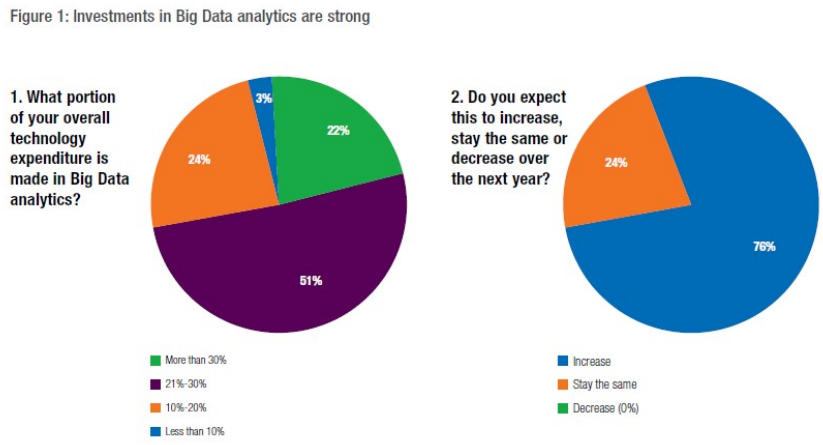


Figura A.2: Interés de las empresas por apostar por el Big Data (2014) [8]

Un campo en expansión puede proporcionarnos un buen terreno para obtener la rentabilidad económica, pero para ello necesitamos contar con alguna ventaja competitiva que nos permita diferenciarnos del competidor. Podemos destacar la modernidad de la tecnología utilizada, que, al estar enfocada al llamado *big data*, consigue realizar tareas que soluciones anteriores no podían resolver en un tiempo prudencial. Además, dentro del tipo de aplicacaciones que ofertan esta posibilidad, podemos destacar de Spark la capacidad de utilizar memoria para agilizar las operaciones.

Finalmente, y en referencia a los servicios sobre los que ejecutar nuestra aplicación, nos encontramos en un punto en el que no es necesario contar con una enorme inversión inicial para hacerse con todo el hardware necesario para conseguir un rápido análisis de datos (clúster con varios nodos, gran capacidad de memoria RAM, etc.). Actualmente, estos servicios pueden contratarse a empresas que ya cuentan con todos estos recursos (tales como Google y su servicio Google Cloud), pagando solo por el uso que hacemos de ellos, lo que abre todavía más el mercado de empresas que podrían permitirse el uso de un clúster para ejecutar este tipo de tareas de minería de datos.

### Clientes objetivo

Aunque, como se ha mencionado, nunca se ha tenido en mente enfocar este producto a ningún aspecto de comercialización concreto, podemos distinguir dos grúpos de clientes sobre los que podría trabajarse:

- **Empresas que manejen grandes volúmenes de datos:** Aunque la minería de datos es algo que parece estar cogiendo fuerza en muchas

áreas, nuestra ventaja competitiva reside en la capacidad para tratar problemas de *big data* (grandes conjuntos de datos) de manera eficiente. Es por ello que nuestro objetivo deberían ser aquellas organizaciones, por lo general de gran tamaño, que tengan el problema de tratar con tal cantidad de datos.

- **Educación e investigación:** Al igual que otras alternativas existentes en el mercado (como Weka [10] o KEEL [2, 1]), nuestro proyecto podría abordar el área de la educación desde una nueva perspectiva: la creación de una librería de algoritmos de minería de datos paralelos.

### Viabilidad legal

En lo que se refiere a aspectos legales, este trabajo no se encuentra en problemática con ningún aspecto que pudiese considerarse de dudosa o nula legalidad, dado que se ha centrado en la implementación y pruebas de diferentes algoritmos.

Con respecto a la minería de datos, el problema más cuestionable es la protección de los datos analizados y la privacidad que se puede ofrecer si dichos datos hacen referencia a una persona o entidad concreta. Sin embargo, en su estado actual, nuestra aplicación no gestiona ni tienen interés en gestionar ningún aspecto de este terreno.

Apéndice $B$

# Especificación de Requisitos

## B.1. Introducción

A lo largo de este apéndice, vamos a hacer referencia a todo lo relacionado con los objetivos del proyecto, indicando apropiadamente las metas y cuáles han sido los requisitos que han guiado el desarrollo de nuestro trabajo.

## B.2. Objetivos generales

Podemos definir dos claros objetivos con los que se comenzó el proyecto:

- La implementación de una serie de algoritmos de selección de instancias que puedan ser ejecutados utilizando el nuevo paradigma de programación paralela que nos ofrece Spark. Estos algoritmos serían *Localy sensitive hashing instance selection* (LSHIS) [4] y *Democratic instance selection* (DemoIS) [9]

- La comparativa entre el rendimiento de las soluciones secuenciales de minería de datos frente a las nuevas soluciones de ejecución paralela que han surgido recientemente. Esta tarea se llevará a cabo utilizando una librería ya existente (Weka) para realizar ejecuciones que sigan el modelo secuencial, y Spark para realizar ejecuciones paralelas.

  Las mediciones deberán están enfocadas en dos sentidos: una comparación general del rendimiento entre Spark y Weka, y la comparación concreta de nuestras implementaciones, definidas en el punto anterior, sobre aquellas que ya estén creadas para la librería Weka.

Igualmente, y conforme el proyecto iba avanzando, se consideró añadir una nueva serie de objetivos, de importancia menor, a nuestra aplicación:

- La creación de una interfaz gráfica más amigable y visual que no solo facilite el uso de todo el material generado, sino que permita, además, definir baterías de experimentos para ser ejecutados al instante o en una máquina diferente.

- El despliegue del proyecto en el entorno donde está pensado para ser ejecutado: un clúster con diferentes nodos y múltiples unidades de procesamiento.

## B.3. Catalogo de requisitos

### Requisitos funcionales

- **RF-1** La aplicación ha de permitir el lanzamiento mediante línea de comandos, de ejecuciones de minería de datos que consten de un algoritmo de selección de instancias y un clasificador.

    - **RF-1.1** Se podrá indicar si se desea o no realizar la medición del tiempo de filtrado.
    - **RF-1.2** Se podrán indicar opciones para la correcta lectura del conjunto de datos. Esto implica indicar donde estará el atributo de clase y si el fichero contiene o no cabecera.
    - **RF-1.3** Se deberá indicar un algoritmo de selección de instancias, junto con su configuración, al iniciar la tarea.
    - **RF-1.4** Se deberá indicar un algoritmo de clasificación, junto con su configuración, al iniciar la tarea.
    - **RF-1.5** Se podrá indicar si se desea realizar validación cruzada en las ejecuciones. En caso de no indicarse nada, existirá una opción por defecto de usar el 10 % del conjunto de datos incial como test para nuestras pruebas.
    - **RF-1.6** Se generará un fichero con el resumen de la ejecución: reducción del algoritmo de selección de instancias, precisión del clasificador y, si ha sido requerido, el tiempo de filtrado.

- **RF-2** La aplicación ha de permitir, vía interfaz gráfica, el lanzamiento de ejecuciones o baterías de ejecuciones que consten de un algoritmo de selección de instancias y un clasificador.

    - **RF-2.1** Se podrán seleccionar algunas de las opciones de lanzamiento de Spark (ruta al directorio de instalación de Spark, ruta al nodo maestro, número de núcleos, número de ejecutores y memoria de cada ejecutor).

  - **RF-2.2** Se podrán elegir uno o más conjuntos de datos, aunque en cada ejecución solo se use uno.
  - **RF-2.3** Se podrá elegir y configurar uno o más algoritmos de preprocesamiento, aunque en cada ejecución solo se use uno.
  - **RF-2.4** Se deberá seleccionar y configurar un algoritmo de clasificación.
  - **RF-2.5** Se podrán configurar las opciones para realizar una validación cruzada.
  - **RF-2.6** Se almacenará un fichero por cada lanzamiento con los datos de reducción y precisión de la ejecución.

- **RF-3** La aplicación ha de permitir, mediante la interfaz gráfica, definir ejecuciones o baterías de ejecuciones y exportar estos experimentos en un archivo .zip con los materiales necesarios.

  - **RF-3.1** Se podrán seleccionar algunas de las opciones de lanzamiento de Spark (ruta al directorio de instalación de Spark, ruta al nodo maestro, número de núcleos, número de ejecutores y memoria de cada ejecutor).
  - **RF-3.2** Se podrán elegir uno o más conjuntos de datos, junto con sus opciones para una correcta lectura de los mismos. Se utilizará un conjunto de datos por ejecución
  - **RF-3.3** Se podrá elegir y configurar uno o más algoritmos de preprocesamiento, usando uno por ejecución.
  - **RF-3.4** Se podrá seleccionar y configurar un algoritmo de clasificación.
  - **RF-3.5** Se deberá generar un archivo .zip que contenga un script para ejecutar la batería de ejecuciones, junto con todos los conjuntos de datos necesarios para las ejecuciones.

- **RF-4** Debe existir una sección destinada a explicar el funcionamiento de la interfaz gráfica.

### Requisitos no funcionales

- **RNF-1** La implementación no ha de tener problemas para correr en paralelo o en sistemas distribuidos formados por varios nodos.
- **RNF-2** El código ha de cumplir con una serie de medidas estáticas de calidad.
- **RNF-3** Los resultados de reducción y precisión generados por nuestros algoritmos de selección de instancias han de ser similares a los generados por la implementación secuencial de dichos algoritmos.

- **RNF-4** La aplicación ha de poder lanzarse en algún de los servicios de computación que ofrecen la posibilidad de correr Spark, concretamente Google Cloud Dataproc.
- **RNF-5** Ha de proporcionarse un sistema que facilite el uso de nuestra aplicación y la generación de baterías de ejecuciones sin necesidad de recurrir a la consola de comandos.

## B.4. Especificación de requisitos

### Diagrama de casos de uso

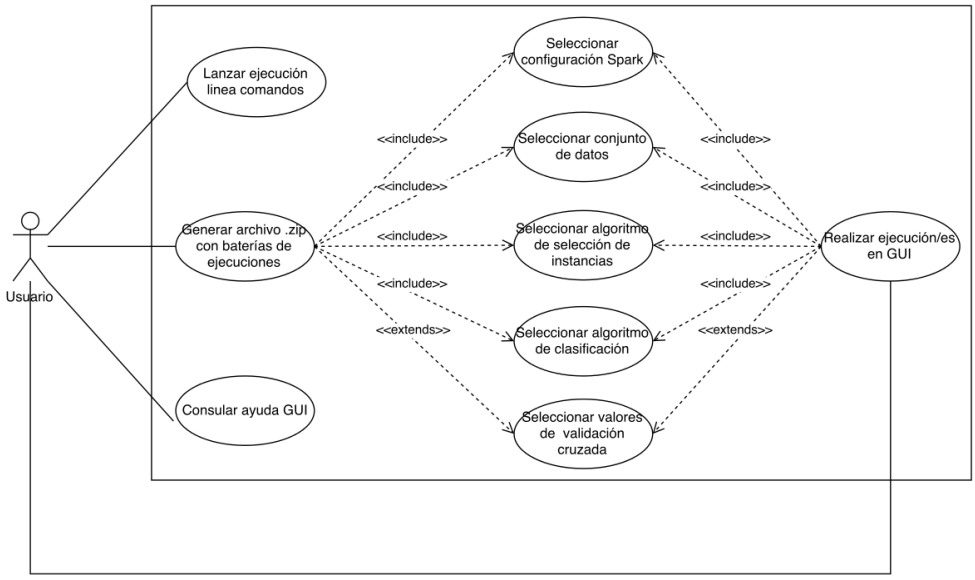


Figura B.1: Diagrama de casos de uso.

### Casos de uso

| | |
|---|---|
| Caso de uso | Lanzar ejecución - Línea de comandos |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-1<br>RF-1.1<br>RF-1.2<br>RF-1.3<br>RF-1.4<br>RF-1.5<br>RF-1.6 |
| Descripción | El usuario puede realizar la invocación, por línea de comandos, de un nueva tarea que implique un selector de instancias y un clasificador. Para ello necesitará aportar, además, una lista de parámetros que permitan la configuración de todos los componentes involucrados. |
| Precondiciones | Ninguna |
| Acciones | 1. El usuario realiza el lanzamiento de la aplicación mediante consola de comandos usando el script "spark-submit" proporcionado por Spark.<br>2. La aplicación realiza la ejecución.<br>2.1 Se muestra periodicamente, en mensajes por consola, el punto de la ejecución en el que nos encontramos.<br>2.2 La aplicación escribe los resultados de la ejecución en un fichero.<br>2.3 La aplicación muestra un último mensaje indicando la correcta ejecución. |
| Postcondiciones | Ningua |
| Excepciones | Los parámetros introducidos en la invocación del programa son erróneos. Se paralizará la ejecución y se informará mediante un mensaje por consola de comandos si este fuese el caso. |
| Importancia | Alta |

Cuadro B.1: Caso de uso "Lanzar ejecución - Línea de comandos"

| | |
|---|---|
| Caso de uso | Realizar ejecución/es GUI |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.1<br>RF-2.2<br>RF-2.3<br>RF-2.4<br>RF-2.5<br>RF-2.6 |
| Descripción | Mediante el uso de la interfaz gráfica, el usuario puede programar una o varias ejecuciones y lanzarlas en el momento. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario completa los campos de opciones comunes de Spark.<br>2. El usuario indica una o varias configuraciones de Spark<br>3. El usuario indica uno o varios conjuntos de datos<br>4. El usuario selecciona uno o varios selectores de instancias.<br>5. El usuario selecciona un clasificador.<br>6. El usuario puede indicar opciones para la validación cruzada (opcional).<br>7. El usuario presiona un botón para ejecutar las tareas programadas.<br>8. La aplicación informa cuando se finalice la operación. |
| Postcondiciones | Los resultados de la ejecución han sido almacenados en un fichero de texto en la ruta correspondiente. |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.2: Caso de uso “Realizar ejecución/es GUI”

| | |
|---|---|
| Caso de uso | Generar archivo .zip con baterías de ejecuciones |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.1<br>RF-2.2<br>RF-2.3<br>RF-2.4<br>RF-2.5<br>RF-2.6 |
| Descripción | Mediante el uso de la interfaz gráfica, el usuario puede programar una o varias ejecuciones y generar un archivo de extensión .zip que contendrá un script de lanzamiento y todos los conjuntos de datos necesarios para la ejecución. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario completa los campos de opciones generales de Spark.<br>2. El usuario indica una o varias configuraciones de Spark<br>3. El usuario indica uno o varios conjuntos de datos<br>4. El usuario selecciona uno o varios selectores de instancias.<br>5. El usuario selecciona un clasificador.<br>6. El usuario puede indicar opciones para la validación cruzada (opcional).<br>7. El usuario presiona un botón para generar el archivo zip.<br>8. La aplicación informa cuando se finalice la operación. |
| Postcondiciones | Se ha generado un archivo de extensión .zip |
| Excepciones | Ninguna |
| Importancia | Media |

Cuadro B.3: Caso de uso "Generar archivo .zip con baterías de ejecuciones"

| | |
|---|---|
| Caso de uso | Seleccionar configuración Spark |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.1<br>RF-3<br>RF-3.1 |
| Descripción | El usuario puede indicar una configuración de Spark rellenando un conjunto de campos de texto. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario ha presionado un botón para añadir una nueva configuración de Spark.<br>2. La aplicación muestra un nuevo diálogo.<br>3. El usuario rellena todos los campos mostrados en el nuevo diálogo.<br>4. El usuario presiona un botón para aceptar la nueva configuración. |
| Postcondiciones | Una nueva configuración ha de haberse añadido a la lista de configuraciones. |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.4: Caso de uso "Seleccionar configuración Spark"

| Caso de uso | Seleccionar conjunto de datos |
|---|---|
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.2<br>RF-3<br>RF-3.2 |
| Descripción | El usuario puede indicar un fichero que contenga un conjunto de datos así como algunos parámetros que permitan su lectura. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario ha presionado un botón para añadir un nuevo conjunto de datos.<br>2. La aplicación muestra un nuevo diálogo.<br>3. El usuario selecciona el botón para buscar en los archivos del sistema un conjunto de datos. Alternativamente, puede escribirlo manualmente.<br>4. El usuario cumplimenta todos los campos restantes mostrados en el diálogo.<br>5. El usuario presiona un botón para aceptar la nueva configuración. |
| Postcondiciones | Un nuevo conjunto de datos, junto con sus opciones de lectura, ha de haberse añadido a la lista de conjuntos de datos. |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.5: Caso de uso "Seleccionar conjunto de datos"

| | |
|---|---|
| Caso de uso | Seleccionar algoritmo de selección de instancias |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.3<br>RF-3<br>RF-3.3 |
| Descripción | El usuario puede seleccionar un algoritmo de selección de instancias de entre las posibles opciones y configurarlo para su ejecución. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1.El usuario ha presionado el botón para añadir un nuevo selector de instancias.<br>2. La aplicación muestra un nuevo diálogo.<br>3. El usuario selecciona, tras pulsar el botón de selección, el filtro que desea.<br>4. La aplicación añade nuevos campos al diálogo. 5. El usuario cumplimenta los nuevos campos mostrados en el diálogo.<br>6. El usuario presiona un botón para aceptar la nueva configuración. |
| Postcondiciones | Un nuevo selector de instancias, junto con su configuración, ha de haberse añadido a la lista de conjuntos de datos. |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.6: Caso de uso “Seleccionar algoritmo de selección de instancias”

| | |
|---|---|
| Caso de uso | Seleccionar algoritmo de clasificación. |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.4<br>RF-3<br>RF-3.4 |
| Descripción | El usuario puede seleccionar un algoritmo de clasificación y configurarlo para su ejecución. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario presiona el botón que permite seleccionar entre los diferentes clasificadores posibles.<br>2. El usuario selecciona un clasificador.<br>3. La aplicación genera nuevos campos con las opciones de configuración del clasificador.<br>4. El usuario rellena los campos con las opciones de configuración. |
| Postcondiciones | Ninguna |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.7: Caso de uso "Seleccionar algoritmo de clasificación"

| | |
|---|---|
| Caso de uso | Seleccionar valores de validación cruzada. |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-2<br>RF-2.5<br>RF-3<br>RF-3.5 |
| Descripción | El usuario puede indicar una configuración de validación cruzada con la que realizar las ejecuciones. |
| Precondiciones | El usuario ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario marca la opción que habilita la validación cruzada.<br>2. El usuario cumplimenta los campos de configuración de la validación cruzada. |
| Postcondiciones | Ninguna |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.8: Caso de uso "Seleccionar valores de validación cruzada"

| | |
|---|---|
| Caso de uso | Consultar ayuda GUI |
| Versión | 1.0 |
| Autor | Alejandro González Rogel |
| Requisitos | RF-4 |
| Descripción | El usuario puede consultar el funcionamiento de la interfaz gráfica mediante un botón de ayuda proporcionado en dicha interfaz. |
| Precondiciones | Se ha iniciado la interfaz gráfica. |
| Acciones | 1. El usuario presiona sobre el botón de ayuda en la interfaz.<br>2. La aplicación muestra una nueva ventana con la ayuda. |
| Postcondiciones | Ninguna |
| Excepciones | Ninguna |
| Importancia | Baja |

Cuadro B.9: Caso de uso "Consultar ayuda GUI"

*Apéndice* $C$

# Pruebas del sistema

## C.1. Introducción

A lo largo de este proyecto, se han realizado una serie de mediciones cuyo objetivo principal consistió en comparar el comportamiento entre ejecuciones secuenciales y paralelas en tareas de minería. A lo largo de esta sección, vamos a mostrar todos los experimentos realizados junto con una descripción de los elementos involucrados y las conclusiones extraídas.

Podemos diferenciar entre tres grandes comparaciones:

- **Comparativa entre las herramientas utilizadas:** Utilizando material ya incluido en las bibliotecas de Weka y Spark, se ha realizado una comparación entre ambas tecnologías. Tendremos en cuenta, no solo el tiempo de ejecución, sino otros factores tales como el uso de memoria o el funcionamiento de los hilos de ejecución (con especial interés en su comportamiento con Spark)

- **Comparativa entre las ejecuciones, en una sola máquina, de los algoritmos LSHIS y DemoIS según sea su implementación secuencial o paralela:** En este caso, nuestro objetivo principal será comparar tiempos de ejecución del algoritmo de filtrado, aunque también tendremos en cuenta parámetros otros parámetros que puedan demostrar el buen funcionamiento de nuestra implementación.

- **Comparativa entre las ejecuciones en un clúster de los algoritmos LSHIS y DemoIS según su implementación secuencial o paralela:** Al igual que en el caso anterior, el objetivo principal en este caso es medir el tiempo de ejecución de la operación de selección de instancias.

| Nombre del conjunto | Instancias | Atributos | Clases |
|---|---|---|---|
| Iris | 150 | 4 | 3 |
| Image Segmentation [15] | 2310 | 19 | 7 |
| Banana shaped [11] | 5300 | 2 | 2 |
| Pen-Based Recognition of Handwritten Digits [15] | 10992 | 16 | 10 |
| Letter Recognition [15] | 20000 | 16 | 26 |
| Human Activity Recognition [21] | 165.632 | 17 | 5 |
| Covertype [15] | 581.012 | 54 | 6 |
| Poker [15] | 1.025.010 | 10 | 10 |
| HIGGS [15] [6] | 11.000.000[1] | 28 | 2 |

Cuadro C.1: Conjuntos de datos utilizados para la comparación entre Weka y Spark.

## C.2. Conjuntos de datos

Los conjuntos de datos (*datasets*), ordenados de menor a mayor según el número total de instancias de cada uno, pueden encontrarse en la tabla C.1. No todos los conjuntos han sido utilizados en todas las comparaciones, puesto que algunas ejecuciones podrían no acabar en un tiempo prudencial, por lo que, en cada prueba, vendrá especificado qué datasets han sido usados.

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

- No existen campos de tipo texto.
- No existen campos vacíos en ninguna de las instancias de los atributos.

Además, es importante indicar que todos los atributos han sido normalizados antes de realizar las mediciones.

[1] El conjunto original consta de 11.000.000 instancias, pero por lo general se he reducido su tamaño original para acercarlo más al tamaño de los otros conjuntos. Cualquier cambio será especificado en la medición a la que afecte.

## C.3. Entorno de las pruebas

Aunque cada medición tiene una serie características únicas que vendrán definidas en la subsección correspondiente, existen algunas circunstancias que son comunes a todas ellas:

- 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 [22].
- Para todas las pruebas hemos usado una validación cruzada 10x2.
- Como ya ha sido mencionado anteriormente, todos los atributos de los conjuntos de datos utilizados han sido normalizados previamente.
- En ejecuciones locales (ver experimento C.4 y C.5) las pruebas se han realizado en un ordenador con capacidad para soportar hasta cuatro hilos simultáneamente y una memoria RAM de 8GB.
- En todas las ejecuciones hemos contado con memoria suficiente como para poder incluir en ella todo el conjunto de datos. Es necesario destacar que, pese a todo, este supuesto no es común dentro del *big data*.

## C.4. Comparativa entre las ejecución de clasificadores en Weka y Spark

El objetivo final de esta comparación es observar el diferente funcionamiento de las librerías, centrando nuestro interés en cómo Spark puede mejorar el tiempo de respuesta a medida que añadimos nuevas unidades de ejecución.

Para realizar las mediciones hemos seleccionado una serie de conjuntos de datos y aplicado sobre ellos un algoritmo de clasificación: 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.

### 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, la ejecución de la validación cruzada y el cálculo del resultado final.
- **Memoria:** Espacio medio de memoria RAM que consume la ejecución del algoritmo.
- **Porcentaje de CPU:** Media del porcentaje de CPU utilizado con respecto a la potencia total. 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.

### Otros aspectos relevantes

- En lo que se refiere a Spark, se ha creado objeto en Scala que es capaz de leer el conjunto de datos, crear diferentes *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.
- Para las ejecuciones en Spark, se ha ejecutado en modo "local", lo que genera en Spark una única unidad (*driver*) a la que se la asignarán todos los recursos que le proporcionemos y que será la encargada de realizar la clasificación.
- Los conjuntos de datos utilizados para esta comparación han sido, ordenados de menor a mayor: "Iris", "Human activity Recognition", "Covertype", "Poker" y "HIGGS". Más información sobre los mismos en la sección C.2.

### Resultados

A continuación se muestran los resultados ordenados de menor a mayor según el tamaño del conjunto de datos:

[2]Los datos de las mediciones sobre el uso de memoria y CPU en el conjunto Iris no son concluyentes debido al corto periodo de tiempo que tardan en ejecutarse.

| Criterio | Weka | Spark 1 hilo | Spark 2 hilos | Spark 4 hilos |
|---|---|---|---|---|
| Tiempo (s) | 0.1 | 1.97 | 2.13 | 2.15 |
| Memoria (MB)[2] | - | - | - | - |
| CPU ( %)[2] | - | - | - | - |

Cuadro C.2: Rendimiento sobre el conjunto de datos Iris.

| 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 C.3: Rendimiento sobre el conjunto de datos Human Activity Recognition.

| 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 C.4: Rendimiento sobre el conjunto de datos Covertype.

| 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 C.5: Rendimiento sobre el conjunto de datos Poker.

| 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 C.6: Rendimiento sobre el conjunto de datos HIGGS (1.469.873 instancias).

| Criterio | Weka | Spark 1 hilo | Spark 2 hilos | Spark 4 hilos |
|---|---|---|---|---|
| Tiempo (s) | 503.93 | 368.72 | 264.02 | 215.21 |
| Memoria (MB) | 1747.22 | 883.28 | 911.77 | 853.58 |
| CPU ( %) | 29 | 26.37 | 50.74 | 91.58 |

Cuadro C.7: 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 observamos en la tabla C.2 o incluso en la tabla C.3, pero su tiempo de ejecución se ve superada por Spark cuando el conjunto de datos empieza a sobrepasar las 100.000. Una evolución de los tiempos de ejecución puede verse en la gráfica C.1

- Nótese en la tabla C.4 que el hecho de que su ejecución sea mayor que la de otros conjuntos de datos más grandes no tiene nada que ver con el comportamiento anómalo de las librerías, sino que tiene un número de atributos mucho mayor que otros datasets mayores y, en el caso de este algoritmo, eso tiene un gran peso sobre la ejecución final. Para más información sobre los conjuntos de datos ver C.2.

- Por lo general, y a la vista de la gráfica C.1, podemos decir que el tiempo de ejecución del algoritmo Naive Bayes de Weka es mayor 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 utiliza menos memoria que Weka para ejecutar el programa. Es probable que esto se deba dos factores importantes de Spark: que las estructuras de datos solo son calculadas cuando se necesitan y que Spark da una gran importancia al correcto uso de los recursos, evitando almacenar nada a no ser que sea especificado en el código.

- 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 y medianos. Ver, por ejemplo, la tabla C.5

- El porcentaje de uso de la CPU en Weka se sitúa siempre entorno a los valores 25-30 %. Esto es así porque la ejecución de todas las tareas es secuencial, consumiendo únicamente un hilo de los 4 que posee la máquina en la que se están realizando las pruebas.

- 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 cuatro. Sin embargo, y como podemos apreciar en las tablas C.5 o C.4, la ejecución con cuatro hilos no aprovecha al máximo las capacidades del procesador cuando el conjunto de datos es pequeño. Observando más de cerca el evento, vemos que, independientemente del número de hilos asignados a Spark, en estos conjuntos de datos únicamente se lanzan dos hilos como máximo. Atribuimos esto a un comportamiento propio de Spark cuando actúa en modo local, que evalúa que no existe necesidad de manejar tantos hilos de ejecución.

  Un ejemplo más ilustrativo de lo anteriormente citado lo encontramos en la imagen C.2, donde se muestran algunos de los hilos de ejecución del lanzamiento de Spark con el algoritmo Naive Bayes sobre el conjunto de datos “Poker” con 4 procesadores asignados. Aunque deberían existir 4 hilos bajo el nombre de *Executor*, solamente se generan dos.

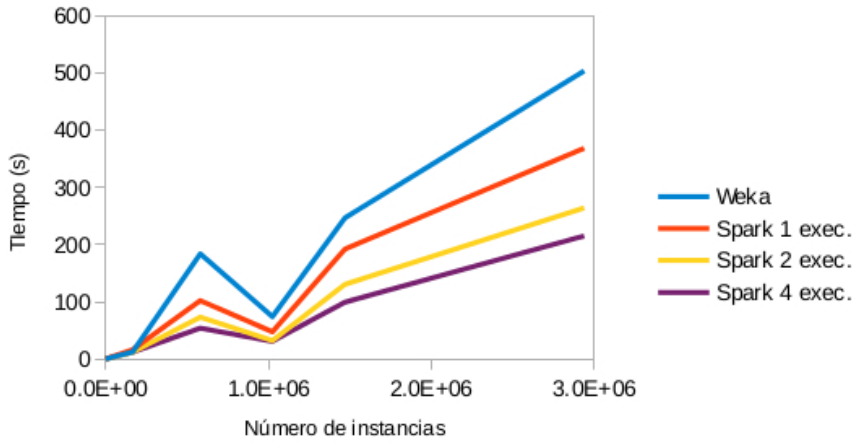


Figura C.1: Evolución del tiempo de clasificación según el número de instancias clasificadas.

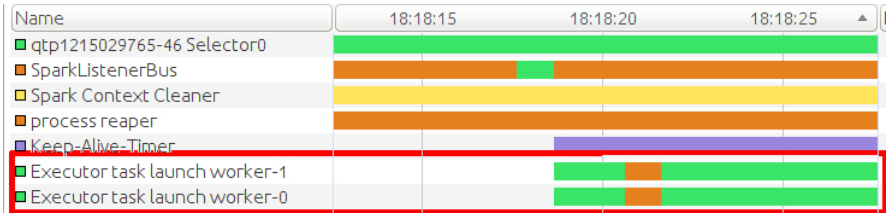


Figura C.2: Hilos de ejecución de Spark en el uso del algoritmo Naive Bayes sobre "Poker" con 4 procesadores asignados.

## C.5. Comparativa entre la ejecución secuencial y paralela, en una sola máquina, de LSHIS y DemoIS

Se procederá a medir diferentes aspectos de la ejecución secuencial y paralela de los algoritmos mencioanados. Para esta comparación, se ha vuelto a utilizar Weka y Spark. En el caso de Weka, ya contábamos con la implementación de los algoritmos [3]. En el caso de Spark, el código utilizado ha sido el resultado de este proyecto.

Mediante estas nuevas pruebas, pretendemos conseguir dos objetivos: Por un lado, queremos evaluar que los algoritmos implementados funcionen como se espera de ellos, es decir, que proporcionen resultados similares a los que proporcionaban los algoritmos ya implementados en Weka. Por otro, queremos realizar un estudio de que índice de mejora hemos conseguido en lo que a

tiempo de ejecución del algoritmo se refiere.

Para todo esto, vamos a realizar lanzamientos de ejecuciones que consten de un algoritmo de selección de instancias (LSHIS o DemoIS) y un clasificador $k$NN que realice una clasificación con los datos obtenidos del filtrado actuando como conjunto de entrenamiento.

De nuevo, la ejecución del algoritmo se analizará tanto en Weka como en Spark, siendo en este último ejecutado con diferente número de ejecutores: 1, 2 y 4. Avanzamos, por lo que se dirá en la sección C.5, que cada ejecutor lleva asignado un hilo de ejecución.

### Criterios comparados

Se ha realizado la comparación teniendo en mente la medición de los siguientes parámetros:

- **Reducción del conjunto de datos:** Relación entre el tamaño del conjunto de datos original y el obtenido tras aplicar el algoritmo. Este valor resulta de la ecuación $1 - (nInstanciasFinales/nInstanciasIniciales)$.
- **Tiempo:** Tiempo real que ha requerido la ejecución del algoritmo de selección de instancias.
- **Precisión de un clasificador con el nuevo conjunto:** Porcentaje de acierto en test al aplicar el algoritmo del vecino más cercano ($k$-Nearest neighbor) sobre el nuevo conjunto de datos.

### Otros aspectos relevantes

- En lo que se refiere únicamente a Spark, se ha corrido en modo *Standalone* [19]. Dentro de este modo, destacar lo siguiente:
  - Solo existe un nodo trabajador (*worker node*) con un solo trabajador (*worker*).
  - Cada ejecutor (*executor*) tendrá asignado un hilo de ejecución.
- Las semillas utilizadas en Spark para generar números aleatorios han sido las mismas en todas sus configuraciones (1,2 y 4 ejecutores.)
- Para las pruebas realizadas con el algoritmo LSHIS se han utilizado los conjuntos de datos, ordenados de menor a mayor: “Letter recognition”, “Human activity recognition”, “Covertype”, “Poker” y “HIGGS”.
- Para las pruebas realizadas con el algoritmo DemoIS se han utilizado los conjuntos de datos, ordenados de menor a mayor: “Image segmentation”, “Banana shaped”, “Pen-Based recognition of handwritten digits” y “Letter Recognition”.

- Para más información sobre los conjuntos de datos ver la sección C.2

### Configuración del algoritmo LSHIS

Durante las mediciones, se ha utilizado la siguiente configuración del algoritmo:

- **Funciones AND:** 10
- **Funciones OR:** 1
- **Anchura de los bucket:** 1

### Configuración del algoritmo DemoIS

Durante las mediciones, se ha utilizado la siguiente configuración del algoritmo:

- **Número de votaciones:** 10
- **Alpha:** 0.75
- **Número de instancias por subconjunto de votación:** Aprox. 1000 instancias.
- **Porcentaje de datos para calcular el límite de votos:** 10 %
- **Vecinos cercanos para calcular el límite de votos:** 1

### Resultados de las mediciones

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 65.86 | 67.03 | 67.2 | 67.27 | - |
| Tiempo (s) | 0.04 | 0.34 | 0.38 | 0.52 | - |
| Precisión ( %) | 92.7 | 92.8 | 93.04 | 92.84 | 95.97 |

Cuadro C.8: Comparativa de LSHIS sobre el conjunto de datos “Letter Recognition”.

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 99.47 | 99.46 | 99.46 | 99.45 | - |
| Tiempo (s) | 0.06 | 0.44 | 0.38 | 0.52 | - |
| Precisión ( %) | 85.61 | 84.39 | 84.94 | 84.68 | 99,57 |

Cuadro C.9: Comparativa de LSHIS sobre el conjunto de datos "Human Activity Recognition".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores |
|---|---|---|---|---|
| Reducción ( %) | 95.8 | 95.61 | 95.6 | 95.6 |
| Tiempo (s) | 0.59 | 3.9 | 2.15 | 2.44 |
| Precisión ( %)* | - | - | - | - |

Cuadro C.10: Comparativa de LSHIS sobre el conjunto de datos "Covertype".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores |
|---|---|---|---|---|
| Reducción ( %) | 63.73 | 61.27 | 61.25 | 61.45 |
| Tiempo (s) | 3.35 | 5.62 | 3.67 | 4.25 |
| Precisión ( %)* | - | | | |

Cuadro C.11: Comparativa de LSHIS sobre el conjunto de datos "Poker".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores |
|---|---|---|---|---|
| Reducción ( %) | 60.91 | 54.46 | 54.47 | - |
| Tiempo (s) | 5.34 | 13.9 | 9.75 | - |
| Precisión ( %)* | - | | | - |

Cuadro C.12: Comparativa de LSHIS sobre el conjunto de datos "HIGGS" (1.469.873 instancias).

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 71.22 | 77.9 | 78.01 | 78.19 | - |
| Tiempo (s) | 5.43 | 4.59 | 3.97 | 4.7 | - |
| Precisión ( %) | 96.32 | 95.9 | 95.46 | 95.51 | 96.76 |

Cuadro C.13: Comparativa de DemoIS sobre el conjunto de datos "Image Segmentation".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 63.63 | 63.97 | 64.1 | 63.35 | - |
| Tiempo (s) | 18.24 | 15.61 | 10.82 | 10.2 | - |
| Precisión ( %) | 85.75 | 85.93 | 85.98 | 86.1 | 87.2 |

Cuadro C.14: Comparativa de DemoIS sobre el conjunto de datos "Banana shape".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 77.12 | 82.58 | 82.05 | 81.3 | - |
| Tiempo (s) | 20.6 | 15.48 | 10.43 | 10.18 | - |
| Precisión ( %) | 98.18 | 98.1 | 97.85 | 98.18 | 99.39 |

Cuadro C.15: Comparativa de DemoIS sobre el conjunto de datos "Pen-Based Recognition of Handwritten Digits".

| Criterio | Weka | Spark 1 Ejecutor | Spark 2 Ejecutores | Spark 4 Ejecutores | Conjunto original |
|---|---|---|---|---|---|
| Reducción ( %) | 56.9 | 56.99 | 56.15 | 58.28 | - |
| Tiempo (s) | 326.98 | 365.9 | 183.3 | 176.26 | - |
| Precisión ( %) | 92.61 | 92.48 | 92.7 | 92.4 | 95.97 |

Cuadro C.16: Comparativa de DemoIS sobre el conjunto de datos "Letter Recognition".

### Aclaraciones

* : El cálculo de la precisión no ha sido medido por la imposibilidad de hacerlo en un tiempo razonable.

### Conclusiones

- En general, puede apreciarse que los valores de reducción y precisión son similares en Weka que en cualquier otra ejecución de Spark. Podemos deducir, por lo tanto, que el comportamiento de estas nuevas implementaciones es bueno, dado que se comportan de la manera que se espera.

- En lo que se refiere al tiempo empleado, podemos apreciar dos comportamientos diferentes dependiendo del algoritmo medido.

  Por su rapidez, incluso en su ejecución secuencial, la ejecución en paralelo del algoritmo LSHIS no aporta ninguna ventaja con respecto a su ejecución secuencial.

  La implementación paralela del algoritmo DemoIS, sin embargo, muestra un mejor rendimiento desde conjuntos de datos muy pequeños, con 2.000 instancias en la tabla C.13. Durante el resto de mediciones hemos conseguido ejecuciones hasta un 50 % más rápidas con Spark (ver C.16)

  Puede observarse la evolución de los tiempos de ejecución de acuerdo al número de instancias analizadas en las gráficas C.4 y C.4.

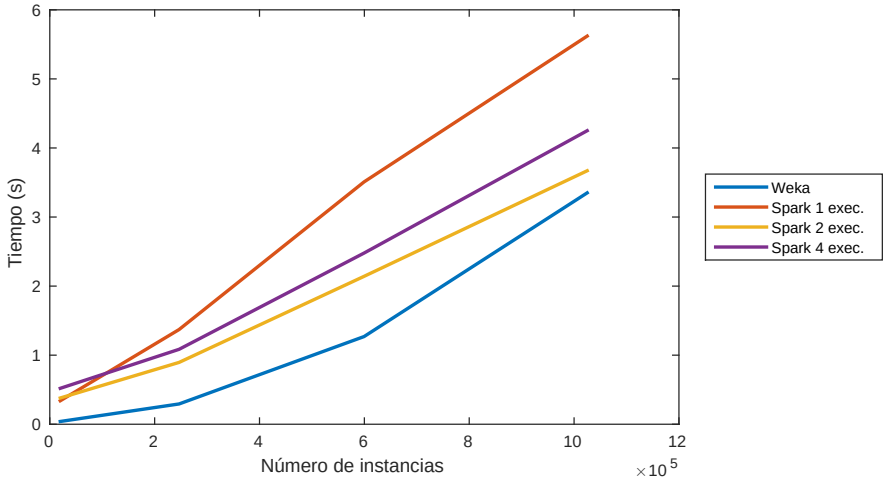


Figura C.3: Evolución del tiempo de ejecución según el tamaño del conjunto de datos usando LSHIS.

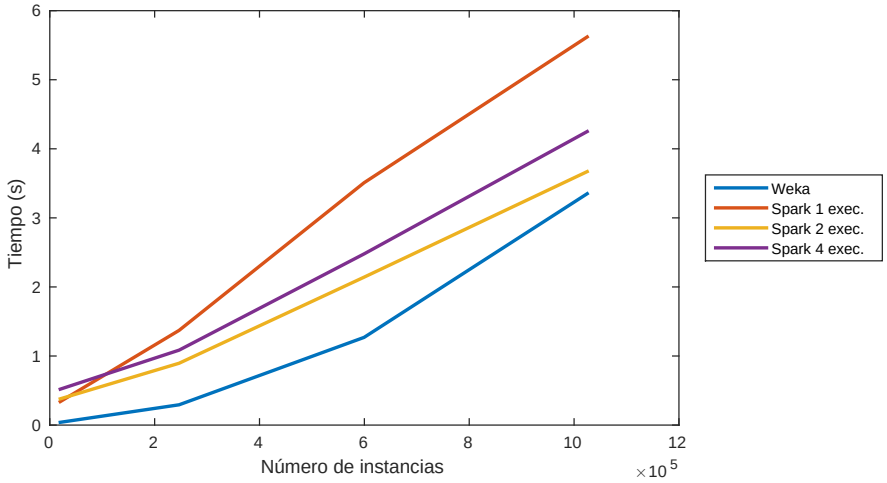


Figura C.4: Evolución del tiempo de ejecución según el tamaño del conjunto de datos usando DemoIS.

- Aunque los resultados de ejecución del algoritmo LSHIS no son mejores que los de su implementación secuencial, dado el supuesto de *Big Data* de que los conjuntos de datos no suelen caber en memoria, la implementación en Spark permite la ejecución del algoritmo cuando la anterior implementación de Weka no lo podría conseguir.

- Puede observarse en cualquiera de las mediciones que, pese a utilizar las mismas semillas para generar números aleatorios, los resultados del algoritmo son ligeramente diferentes según qué configuración de Spark

usemos. Esto se debe a la ejecución paralela no garantiza un orden en el que los datos pueden ser computados o distribuidos, por lo que la ejecución se ve ligeramente afectada.

- Con los conjuntos de datos utilizados, la ejecución con cuatro ejecutores no parece que aporte una gran ventaja. En el caso del algoritmo LSHIS, probablemente debido al exceso de comunicación entre ejecutores, añadir más de dos ejecutores resulta incluso perjudicial.

- Aunque no puede apreciarse fácilmente en los resultados mostrados, cuando solo existe una función OR, el porcentaje de reducción del algoritmo LSHIS depende fuertemente de la semilla utilizada para generar las funciones hash. De hecho, en conjuntos de datos como HIGGS (ver C.2) han llegado a apreciarse ejecuciones cuyo resultado variaba en hasta 200.000 instancias con tan solo modificar la semilla. Es por ello que en algunas mediciones, como en tabla C.8 y tabla C.12, pueden apreciarse porcentajes de reducción diferentes entre las mediciones realizadas con Weka y con Spark.

- Puede observarse como en algunas mediciones del algoritmo DemoIS (tabla C.13 y tabla C.15) el porcentaje de reducción es mayor con Spark que con Weka, sin afectar esto a la precisión. Esto se debe a una ligera variación en el cálculo del *fitness* de la nueva implementaciones, donde utilizamos un subconjunto de test más parecido al original. Un ligero cambio en los parámetros de lanzamiento de Weka (parámetro *alpha*) conduciría al mismo resultado obtenido con Spark.

## C.6. Medición del tiempo de filtrado, en un clúster, de LSHIS y DemoIS

En un intento de aumentar los recursos disponibles y de probar el funcionamiento en un sistema distribuido, se han lanzado pruebas sobre un servicio clúster. Se intenta, no solo realizar ejecuciones que necesiten de más recursos, sino también ver el funcionamiento de nuestro programa en un entorno donde encontramos problemas adicionales, como puede ser un aumento del tiempo de comunicación entre los diferentes nodos (hasta ahora, en la medición anterior C.5, solo existía un único nodo)

El servicio utilizado, contratado por la Universidad de Burgos, es un servidor Cluster Dell con 63 nodos 4xQuadCore Xeon @ 2 GHz y un total de 252 núcleos. Está configurado con un sistema de gestión de colas de trabajo PBS (*Portable Batch System*), por lo que para la ejecución de Spark se ha necesitado de un script de lanzamiento proporcionado por la Universidad de Ohio [5].

En esta ocasión, se han lanzado ejecuciones con Weka y en Spark con hasta 16 procesadores.

Sin embargo, las ejecuciones tuvieron que ser suspendidas por problemas entre el script mencionado y el sistema de gestión de colas, que generaban que ciertos núcleos dejaran trabajar y paralizasen la ejecución hasta que volvían a ser iniciados. Es por ello que no contamos con todas las mediciones que se plantearon en un primer momento y que no podemos asegurar que los tiempos medidos sean completamente veraces, por lo que no han sido incluidos en la memoria ni realizaremos un análisis sobre ellos.

Si puede decirse, sin embargo, que nuestros algoritmos han sido ejecutados en otro servicio (Google Cloud Dataproc) con conjuntos de datos mayores que los que ocasionaban problemas en el servidor Clúster Dell y no se han detectado problemas en la ejecución, por lo que se descarta que pueda haber un error en el código de la aplicación.

*Apéndice D*

# Documentación de usuario

## D.1. Introducción

A lo largo de este apéndice vamos a indicar el material necesario para poder ejecutar el proyecto, así como la manera de instalarlo y hacerlo funcionar una vez contemos con todos los elementos requeridos.

## D.2. Requisitos de usuarios

### Requisitos software

A continuación se muestra una lista de todos los requisitos mínimos para la ejecución del proyecto. En las secciones D.3 y D.4 podrá encontrarse el método de instalación para cada uno de estos componentes.

Podemos advertir diferentes distribuciones (ver sección D.5) y diferentes maneras de ejecutar el proyecto (ver sección D.5), por lo que no todos los componentes son necesarios en todos los casos

- Java 8
- Apache Maven (Solo necesario si requerimos Apache Spark)
- Apache Spark (Solo necesario si deseamos lanzar las ejecuciones en nuestra máquina)

### Requisitos mínimos del sistema

Dado que existen diferentes modos de uso del proyecto, es necesario indicar aquí varias configuraciones:

- Únicamente deseamos generar un archivo .zip para nuestro proyecto:

  - **Procesador:** Pentium 2 266 MHz [16]
  - **Memoria RAM:** 128MB [16]
  - **Espacio libre en disco:** aprox. 180 MB + Espacio adicional para los conjuntos de datos.

- Deseamos ejecutar los algoritmos en nuestro proyecto:

  - **Procesador:** Procesador multihilo con capacidad de soportar, al menos, 2 hilos simultaneamente.
  - **Memoria RAM:** 4 GB
  - **Espacio libre en disco:** aprox 2.1GB más el espacio adicional para los conjuntos de datos que deseemos almacenar.

Quede constancia, además de los requisitos mencionados, que hay que tener en cuenta dos aspectos más:

- La memoria mínima que un ejecutor de Spark puede tener es 1 GB. Por lo tanto, tal vez sea necesario ajustar el número de ejecutores si no tenemos memoria suficiente.
- Los requisitos definidos para la ejecución de Spark en ningún caso se van a parecer a los usados en una ejecución "real" en un clúster, cuyos requisitos recomendados pueden encontrarse en la página oficial de Spark [18].

## D.3. Instalación

A continuación se van a definir los métodos de instalación de todos los componentes necesarios para ejecutar el proyecto, de cara a facilitar el mantenimiento en futuras versiones de la biblioteca y permitir la instalación al usuario.

Estos métodos de instalación hacen referencia a la instalación en máquinas con un sistema operativo basado en alguna distribución de Debian. En nuestro caso, esta distribución ha sido Ubuntu 14.04.

### Oracle Java 8

Aunque existen distribuciones libres de Java, usaremos la que proporciona Oracle (`http://www.java.com`) por el hecho de que incluye una serie de programas que pueden usarse para medir el rendimiento de aplicaciones Java (JConsole y JVisualVM). Cualquier otra distribución será igualmente válida, pero es posible que no incluya estas herramientas. Podemos acceder tanto

al JRE como al JDK desde la siguiente dirección: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Sin embargo, Oracle no proporciona una instalación automática de Java para Linux. Es por esta razón vamos a utilizar un PPA (*Personal Package Archive*) que proporciona un instalador para diferentes versiones de Java [20]. Este instalador no contiene ningún archivo Java, pero será el encargado de descargarlos en nuestra máquina (de igual manera que podríamos hacerlo manualmente) y realizar la instalación automáticamente, facilitando el proceso de instalación.

Una vez entendido esto, ejecutaremos los siguientes comandos:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
$ sudo apt-get install oracle-java8-set-default
```

Podemos comprobar si la instalación ha sido correcta ejecutando el comando *java -version* en la terminal. Si todo ha salido bien deberíamos recibir una salida similar a la siguiente:

```
$ java -version
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b17,
mixed mode)
```

### ISAlgorithms

Puede ser descargado desde el repositorio del proyecto desde https://bitbucket.org/agr00095/tfg-alg.-seleccion-instancias-spark.

Para más información sobre las posibles distribuciones ver la sección D.5.

Aunque para algunas acciones no será necesario que el paquete se encuentre en un directorio concreto ni que tenga un nombre específico, se recomienda situar el fichero en el lugar de instalación de Spark, dentro de una carpeta llamada "lib". Así pues, en nuestro caso la ruta donde situar el fichero sería *$SPARK_HOME/lib*. Igualmente se recomienda renombrar el archivo con el nombre "ISAlgorithms.jar".

## D.4. Instalaciones Opcionales

Existen una serie de componentes que no van a ser necesarios en todas las distribuciones del producto (ver D.5). Los componentes definidos a conti-

nuación son necesarios para permitir la ejecución de tareas de minería en la máquina local.

### Apache Maven

Este elemento, si bien no es necesario para la ejecución del proyecto, es necesario para la construcción de Apache Spark (ver D.4), por lo que es incluido en este listado.

Lo primero que debemos hacer es descargarnos el paquete que contiene la herramienta. Esto podemos hacerlo desde el navegador en la página oficial de Apache Maven (https://maven.apache.org/) o mediante el siguiente comando en consola:

```
$ wget http://apache.rediris.es/maven/maven-3/3.3.3
  /binaries/apache-maven-3.3.3-bin.tar.gz
```

Recordar que el código de arriba es orientativo, pudiéndose seleccionar otro lugar desde donde realizar la descarga u otra versión del producto.

Descomprimimos el paquete descargado y lo movemos a la carpeta */usr/local*:

```
$ tar -zxf apache-maven-3.3.3-bin.tar.gz
$ sudo mv apache-maven-3.3.3 /usr/local
$ sudo ln -s /usr/local/apache-maven-3.3.3/bin/mvn
  /usr/bin/mvn
```

Podemos comprobar que la instalación ha sido realizada correctamente si al escribir el comando *mvn --version* recibimos una salida parecida a la siguiente:

```
$ mvn --version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0df;
  2015-04-22T13:57:37+02:00)
Maven home: /usr/local/apache-maven-3.3.3
Java version: 1.8.0_66, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-oracle/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.19.0-25-generic", arch:
  "amd64", family: "unix"
```

### Apache Spark

Nos dirigiremos a la página oficial de Apache Spark (http://spark.apache.org/). Elegiremos la versión 1.5.1 por ser la utilizada a lo largo de la práctica, y descargaremos el código fuente. Igualmente, y como hemos mencionado en

otras instalaciones, podemos ejecutar el siguiente comando para hacernos con el paquete:

```
$ wget http://apache.rediris.es/spark/spark-1.5.1
  /spark-1.5.1.tgz
```

Una vez tengamos el archivo en nuestra máquina lo descomprimimos y movemos a la carpeta que deseemos lanzando estos comandos desde el directorio que contenga el paquete descargado:

```
$ tar -xvf spark-1.5.1.tgz
$ sudo mv spark-1.5.1 /opt
```

Finalmente vamos a construir Spark utilizando los siguientes comandos desde la carpeta donde lo hemos ubicado. En nuestro caso, estará en */opt/spark-1.5.1*:

```
$ sudo ./dev/change-scala-version.sh 2.11
$ sudo mvn -Dscala-2.11 -Pnetlib-lgpl \
  -DskipTests clean package
```

Es importante incluir la opción `-Pnetlib-lgpl` para que Spark incluya una serie de clases utilizadas por su librería MLlib[17]. Igualmene, destacar que esta invocación de Spark no vincula la librería con Hadoop o cualquier administrador de clústeres posible, por lo que si se requiere alguno de estos servicios deberán añadirse nuevos parámetros al comando, algo que no se discutirá aquí.

Esta última operación llevará un tiempo.

Finalmente, si deseamos comprobar que Spark se ha desplegado correctamente podemos ejecutar, por ejemplo, el intérprete de comandos de Scala para Spark:

```
$ ./bin/spark-shell
```

Aunque recibiremos una serie de avisos (*warnings*) debido a que no hemos configurado ciertos aspectos de Spark, el intérprete debería poder lanzarse y funcionar sin problemas.

Finalmente, sería recomendable que, por comodidad, se añadiese la variable $SPARK_HOME al sistema. Para ello, podemos modificar el fichero */.profile* y añadirle la siguiente línea:

```
export SPARK_HOME="/opt/spark-1.5.1/
```

Para que este cambio tenga efecto necesitaremos, sin embargo, reiniciar la sesión de usuario.

## D.5. Manual del usuario

Existen multitud de formas de lanzar una ejecución con nuestra aplicación, por lo que se dedicará esta sección a hablar y definir cada una de ellas.

Brevemente, vamos a considerar 4 formas de lanzamiento:

- Lanzamiento desde línea de comandos.
- Lanzamiento desde la interfaz gráfica.
- Definición de una batería de ejecución en la interfaz gráfica y lanzamiento por separado.
- Lanzamiento en el servicio de computación en la nube Google Cloud Dataproc.

Recordemos, antes de empezar, que nuestra aplicación consta de varias distribuciones y no todas pueden lanzar experimentos de todas las maneas posibles. Para ello, conviene revisar primero la sección D.5.

Igualmente, existen diferentes modos de despliegue de Spark, algo que también influye en nuestra ejecución. Es por ello recomendable leer primero la subsección D.5, independientemente de la manera en la que deseemos ejecutar la aplicación.

### Distribuciones

Hemos de entender que existen tres distribuciones diferentes de nuestra aplicación, cada una de ellas con unas características diferentes en función de la finalidad del producto, por lo que es necesario elegir aquella que se adapte mejor al uso que pensemos dar al proyecto:

- **ISAlgorithms:** Presenta todo el material del proyecto, desde la interfaz gráfica hasta los algoritmos, pasando por todo el material necesario para poder lanzar ejecuciones desde Spark. Es un tipo de distribución general que puede realizar todas las operaciones posibles. La versión de Scala utilizada para compilar las clases ha sido 2.11.7.
- **ISAlgorithms_gui:** Contiene únicamente los componentes de la interfaz gráfica, por lo que no puede lanzar ejecuciones. Aun así, permite definir experimentos y generar un archivo .zip para exportarlos a cualquier otra máquina que contenga una distribución del programa que permita el lanzamiento. La versión de Scala utilizada durante la compilación ha sido 2.11.7.

- **ISAlgorithms_cluster:** Contiene todo el material a excepción de la interfaz gráfica, que se encuentra innecesaria al ser una red de nodos remota el objetivo de esta distribución. La versión de Scala utilizada para compilar ha sido 2.10.6.

### Modos de despliegue de Spark

Spark proporciona diferentes modos de lanzamiento, que vamos a diferenciar en tres grupos:

- **Modo local:** El más simple de todos. Como su propio nombre puede dejar ver, en este modo solo contaremos con un único trabajador que tendrá asignados todos los recursos. A nivel interno, las aplicaciones locales no distribuyen el trabajo a unidades llamadas *executors*, como se hace en cualquier otro tipo de despliegue, sino que todo el trabajo es manejado por una sola unidad denominada *driver*.

  No necesita ningún tipo de gestión previa a ser desplegado, solo ha de indicarse que se desea correr en este modo cuando lanzamos una aplicación con Spark.

  Es una buena opción cuando se están realizando pruebas pero está limitado en muchos aspectos.

- **Modo Standalone:** Proporciona un tipo de despliegue sin tener que depender de software de administración de clústeres, aunque requiere una versión compilada de Spark en cada nodo.

  Puede resultar eficiente usarlo en un grupo de nodos pequeño, pero no es buena opción cuando contamos con una red más amplia [14].

  Este es el modo de despliegue que hemos utilizado en gran parte del proyecto. Una explicación más detallada de como iniciarlo en una sola máquina (el ordenador en el que se esté trabajando) puede encontrarse en la subsección D.5.

- **Otros modos de despliegue**: Existen numerosos administradores de clústeres que pueden funcionar con Spark, tales como Mesos o YARN, pero en ningún caso hemos necesitado programar un servicio similar, por lo que no entraremos a detalle. Cabe destacar, sin embargo, que el gestor de clústeres YARN es el que usa Google Cloud Dataproc cuando hemos lanzado ejecuciones sobre él.

### Iniciar el modo Standalone en una máquina (Opcional)

Es posible que, intentando evitar el modo de ejecución de Spark local, queramos ejecutar nuestros programas como si funcionasen en un auténtico

clúster, aunque éste solo tenga un nodo. Para ello son necesarios algunos pasos previos.

Desde el directorio de instalación de Spark podemos ejecutar el siguiente comando para iniciar el nodo maestro:

```
./sbin/start-master.sh
```

Esta operación, que podría tardar varios segundos, debería abrir un nuevo puerto (http://localhost:8080/), desde donde poder acceder a la información de nuestra red de nodos que, de momento, debería estar vacía.

Ahora, vamos a iniciar un nodo trabajador (*worker*) y asociarlo al master con el siguiente comando:

```
./sbin/start-slave.sh <master-spark-URL>
```

Si hemos los pasos correctamente, y no hemos modificado la configuración por defecto, la dirección URL del nodo master debería ser algo similar a *spark://ruta:puerto*, donde "ruta" es el atributo $HOSTNAME de la máquina donde hemos iniciado el nodo maestro y "puerto", por defecto, es 7077. En caso de duda, siempre podemos visitar `http://localhost:8080/`, donde ,entre otra información, podemos ver la URL del master.

### Lanzamiento de ejecución desde línea de comandos

Es el método de ejecución más rápido pero requiere un mínimo conocimiento sobre los algoritmos y parámetros que podemos configurar.

Se presenta un ejemplo de ejecución en el código D.1. Sobre este ejemplo, vamos a intentar comprender cada uno de sus componentes de la llamada para poder, en el futuro, configurar nuestras propias ejecuciones.

```
$SPARK_HOME/bin/spark-submit \
--master spark://alejandro:7077 \
--class "launcher.ExperimentLauncher" \
$SPARK_HOME/lib/ISAlgorithms.jar ISClassExec -r \
 ./Datasets/Banana-5k/banana_norm.csv -f \
instanceSelection.demoIS.DemoIS -np 5 \
-c classification.seq.knn.KNN -k 1 -cv 10 1
```

Código D.1: Ejemplo de ejecución desde línea de comandos

- **Script de lanzamiento:** *$SPARK_HOME/bin/spark-submit* es un código que proporciona Spark para permitir el lanzamiento de aplicaciones. Siempre vamos a necesitar esta sentencia al lanzar una aplicación.

- **Opciones de Spark:** Spark cuenta con multitud de parámetros para personalizar su funcionamiento, pero solo mencionaremos algunos que nos son de especial interés:

  - **Nodo master:** *--master spark://alejandro:7077* Indica la ruta al nodo maestro de nuestro clúster. Esta opción puede tomar como parámetro la sentencia *local[n]*, para indicar una ejecución a nivel local con *n* número de núcleos o la sentencia *spark://ruta:puerto* si la ejecución se realiza en otro modo de despliegue de Spark.
    Puede definirse una ruta por defecto en los ficheros de configuración de Spark (concretamente en *$SPARK_HOME/conf/spark-defaults.conf*), pero se le hace una mención especial porque, sin este argumento definido en algún sitio, la ejecución no podría tener lugar.
  - **Clase principal:** *--class "launcher.ExperimentLauncher"* señala la clase principal a cargar dentro del .jar que deseamos lanzar y que mencionaremos más adelante.

  Todas las diferentes opciones que pueden consultarse mediante el comando *$SPARK_HOME/bin/spark-submit --help* o haber sido previamente definidas en los ficheros de configuración de Spark.

- **Archivo .jar:** *$SPARK_HOME/lib/ISAlgorithms.jar* define el archivo .jar que deseamos ejecutar. Si se han seguido los pasos de instalación previos, debería encontrarse en la misma ruta que la del ejemplo.

- **Argumentos para nuestro programa:** El resto de la sentencia consta de valores para la configuración del lanzamiento de nuestra aplicación. Podemos encontrar un listado con todos los parámetros disponibles en la sección D.5. Aunque esta parte del programa es la que más variaciones puede sufrir de una ejecución a otra, se puede definir una estructura general:

  - *Tipo de ejecución (obligatorio):* El primer argumento indica la tarea de minería de datos que deseamos ejecutar.
  - **Argumentos para el lector de conjuntos de datos (obligatorio):** Comienza con la sentencia "-r" y debe indicar la ruta al fichero que contenga el conjunto de datos seguido de las posibles necesarias para la correcta lectura del fichero.
  - **Argumentos para la configuración del algoritmo de selección de instancias (obligatorio):** Comienza con la sentencia "-f" y debe indicar la ruta, dentro del fichero .jar, a la clase que contenga el algoritmo que deseamos lanzar y, seguidamente, una lista con las opciones de configuración de dicho algoritmo en el caso de que queramos cambiar las opciones por defecto.

- **Argumentos para la configuración del algoritmo de clasificación (obligatorio):** Comienza con la sentencia "-c" seguida de la ruta, dentro del fichero .jar, a la clase que contenga el algoritmo que deseamos lanzar y una lista con opciones para configurar los valores por defecto del algoritmo.
- **Argumentos para la configuración de la validación cruzada (opcional):** Empieza con la sentencia "-cv" seguida de uno o dos números, siendo el primero el que indica el número de *folds* que deseamos crear y el segundo una semilla para la división aleatoria del conjunto de datos inicial. Si no indicamos ningún parámetro, se ejecutará un experimento donde el conjunto de entrenamiento supondrá un 90 % del total de instancias y, el de test, un 10 %.

Si todo ha salido correctamente, la ejecución empezará a procesarse. Es posible que veamos aparecer algunos mensajes informativos por consola, dependiendo de si hemos habilitado esa opción en Spark o hemos mantenido la que se ofrece por defecto.

Cuando la ejecución haya finalizado correctamente, se habrá generado una nueva carpeta de nombre "results" en el directorio desde el cual lanzamos la ejecución. Dentro del mismo, podremos encontrar un archivo con el nombre del clasificador usado, seguido de la fecha y hora de la finalización. Accediendo a él podremos ver los aspectos medidos de la ejecución.

### Lanzamiento de ejecución o batería de ejecuciones desde la interfaz gráfica

Antes de empezar, merece la pena recordar que para llevar a cabo este paso es necesario que, tal y como se ha indicado en D.3, el fichero .jar que vamos a utilizar debe estar situado en la ruta *$SPARK_HOME/lib*.

El primer paso consistirá en lanzar la interfaz gráfica. Para ello, podemos hacer doble clic sobre el archivo .jar distribuido o ejecutar la siguiente sentencia en la línea de comandos:

```
$ java -cp target/ISAlgorithms.jar gui.SparkISGUI
```

De cualquiera de las maneras, tendremos a nosotros una nueva ventana similar a la que se muestra en la imagen D.1. Pueden observarse cuatro áreas principales en el cuerpo de la aplicación, junto un menú superior y otro inferior.

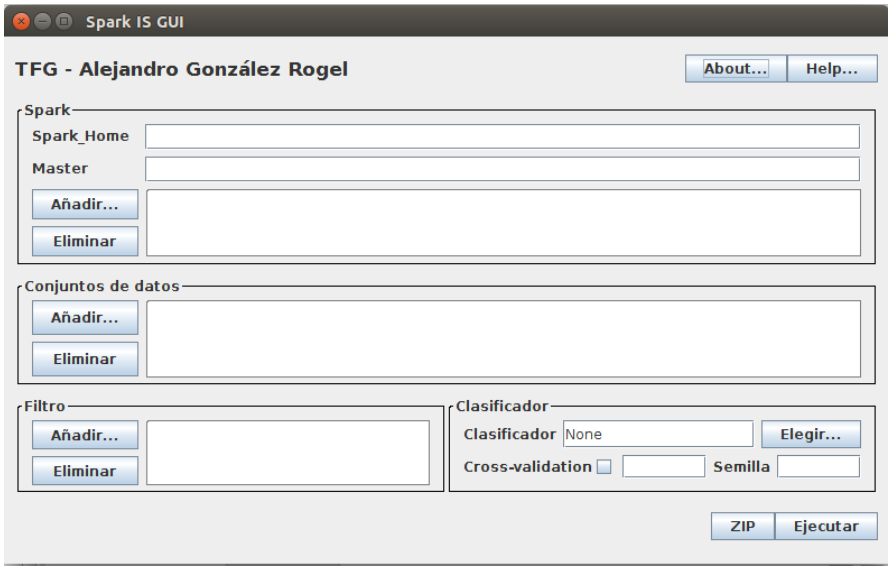


Figura D.1: Ventana principal de la interfaz gráfica.

- **Menú superior:** Contiene el título del proyecto y dos botones de carácter informativo. Si no comprendemos el funcionamiento de la interfaz, sería útil consultar el botón de ayuda de la propia interfaz, donde se realiza una explicación más detallada de cada uno de los campos existentes en la interfaz.

- **Cuerpo:** Se encuentra subdividido en cuatro grandes bloques, cada uno de ellos dedicado a un aspecto diferente del lanzamiento. Podemos apreciar una estructura similar en la mayoría de ellos, que contienen un listado donde aparecerán todas las configuraciones que se hayan indicado hasta el momento, así como un par de botones para añadir o eliminar nuevas configuraciones.

    - **Panel de configuración de Spark:** Nos permitirá seleccionar una o varias opciones de configuración para iniciar nuestra tarea en Spark. Algunas de estas opciones son comunes para todas las tareas que programemos, tales como "SPARK_HOME" y "Master". Otras, variarán en cada experimento pueden añadirse o eliminarse pulsando sobre los botones "Añadir..." y "Eliminar" respectivamente.

      El botón "Añadir..." dará paso a un nuevo diálogo (ver imagen D.2) donde rellenar una serie de parámetros, ninguno de los cuales debería dejarse vacío. Una vez indicados todos los parámetros, aceptar esa configuración resultará en una nueva línea en la tabla de configuración de la pantalla principal.

El botón "Eliminar" suprime la fila de la tabla seleccionada, si es que hubiese alguna.

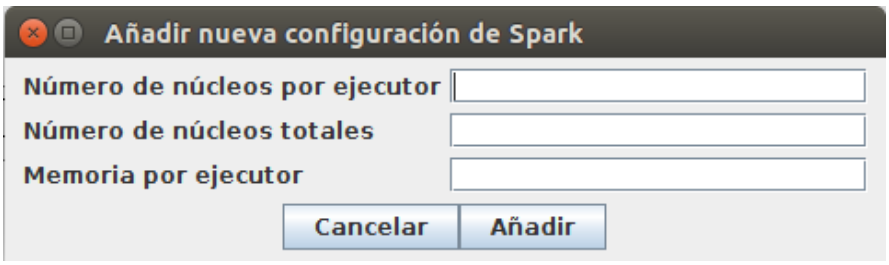


Figura D.2: Diálogo para la selección de opciones de Spark.

- **Panel de selección de conjunto de datos:** Cuenta con un listado de los conjuntos añadidos hasta el momento y una serie de botones que permiten añadir/eliminar los conjuntos de datos seleccionados para el experimento.

  Al igual que en el caso anterior, pulsar sobre el botón "Añadir..." generará un diálogo (ver imagen D.3)que permitirá indicar un nuevo conjunto de datos y las opciones de lectura necesarias, si es que las hubiese.

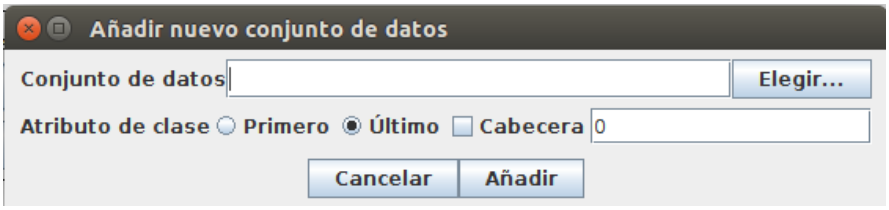


Figura D.3: Diálogo para la selección de un conjunto de datos.

- **Panel de selección de filtro:** Similar al panel anterior, muestra un listado de todas las configuraciones propuestas hasta la fecha y la posibilidad de añadir más o eliminar las ya existentes.

  Mencionar una peculiaridad en este caso. Al presionar sobre el botón "Añadir..." se abrirá un panel con solo un campo para seleccionar un algoritmo selector de instancias. El resto de campos se generarán de manera automática cuando seleccionemos un algoritmo concreto (ver imágenes D.4 y D.5).
- **Panel de selección de clasificador y validación cruzada:** Su funcionalidad es similar a la descrita en paneles anteriores, concretamente el panel de selección de filtro. Sin embargo, es este caso solo es posible añadir un único clasificador (así como solo una única configuración de validación cruzada), por lo que las opciones que

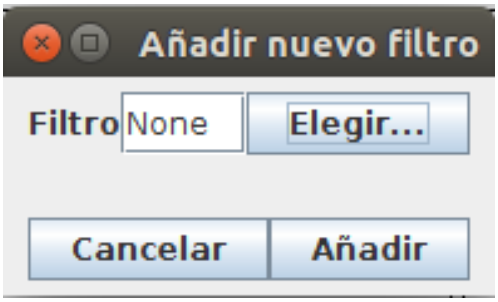


Figura D.4: Diálogo para la selección de filtro.

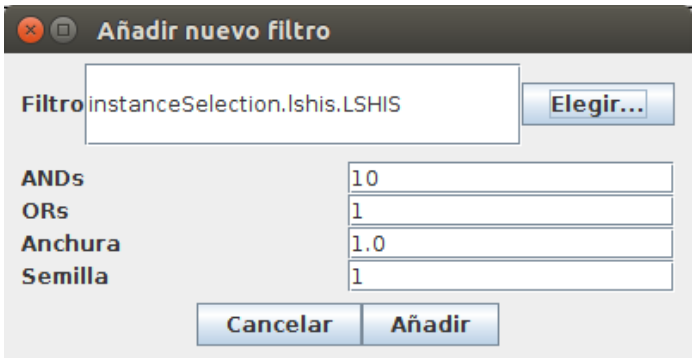


Figura D.5: Diálogo para la selección de filtro después de seleccionar un algoritmo.

típicamente se mostraban en un diálogo separado, ahora se muestran en la propia pantalla (ver imagen **??**).

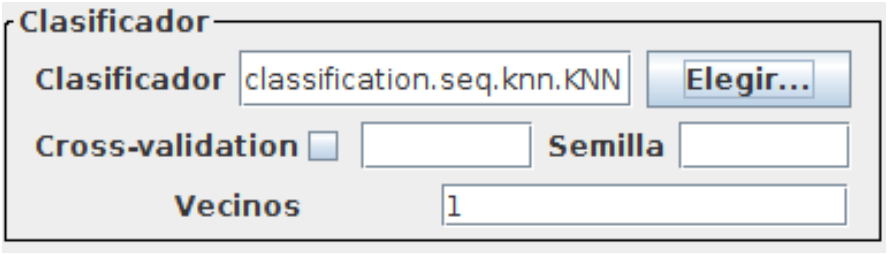


Figura D.6: Panel para la configuración del clasificador y la validación cruzada.

- **Menú inferior:** Cuenta con dos botones para ejecutar diferentes acciones. La modalidad "ZIP" será descrita en la subsección D.5, pero en esta sección nos interesa presionar el botón "Ejecutar".

Antes de lanzar una ejecución surge la pregunta de cuantas ejecuciones

diferentes se han programado. Se generarán tantas ejecuciones como posibles combinaciones entre componentes se puedan crear con las configuraciones propuestas. Así pues, si, por ejemplo, se han indicado dos configuraciones de Spark, dos conjuntos de datos y dos filtros se realizarán $2 \times 2 \times 2 = 8$ ejecuciones, puesto que el clasificador será siempre el mismo.

Una vez hemos cumplimentado todos los campos requeridos (los únicos opcionales son los que se refieren a la validación cruzada) y presionamos sobre el botón "Ejecutar" veremos como en la esquina inferior izquierda se muestra un nuevo texto indicando la ejecución que se está realizando. Un ejemplo de un experimento completamente configurado y en funcionamiento puede verse en la imagen D.7

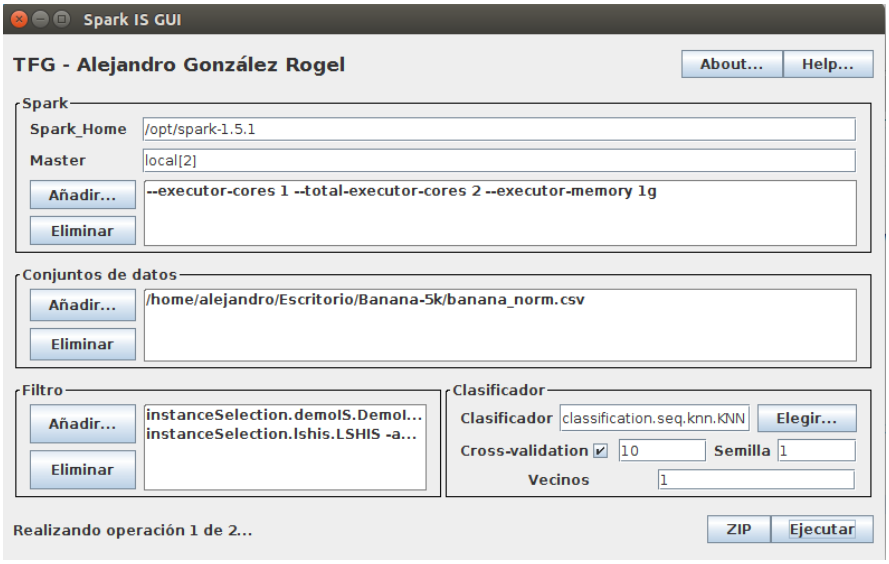


Figura D.7: Ventana principal de la interfaz gráfica rellena.

No se pueden lanzar varias baterías de experimentos a la vez, así que si intentásemos crear una nueva ejecución y ejecutarla mientras la anterior continúa funcionando recibiremos un mensaje de error.

Una vez terminadas todas las ejecuciones, la interfaz nos informará de ello mediante un nuevo diálogo. Podemos ir a consultar el resultado en el directorio *results* que se ha generado en la carpeta desde la que hemos lanzado la ejecución. Si hemos iniciado el programa haciendo doble clic sobre el fichero .jar este directorio probablemente se encuentre en la carpeta $HOME del usuario.

### Definición de una batería de ejecución y lanzamiento por separado

El procedimiento inicial es similar al marcado en D.5, es decir, una vez abierta la interfaz gráfica hemos de indicar todas las configuraciones de ejecución que deseemos. La diferencia esta vez la encontramos al final de la operación, donde en lugar de presionar sobre el botón "Ejecutar", lo haremos sobre el botón "ZIP"

Esta operación, que tal vez puede tardar unos instantes si hemos seleccionado conjuntos de datos muy grandes, generará una carpeta de nombre "zip" en el directorio desde donde hemos invocado la interfaz gráfica. Dentro de la carpeta encontraremos un nuevo archivo .zip que contendrá un script de ejecución .sh junto con todos los conjuntos de datos que necesitemos para realizar las ejecuciones que hemos definido previamente.

Ese fichero lo moveremos a donde sea necesario, típicamente un servicio clúster donde se encuentre Spark instalado junto con nuestra librería en el directorio *$SPARK_HOME/lib*, tal y como se ha indicado en D.3.

Una vez allí, el archivo puede descomprimirse usando el comando:

```
$ unzip <fichero-zip>
```

Es posible que, para descomprimir el archivo necesitemos un paquete *unzip* instalado. Si no existe, podremos obtenerlo con el comando

```
$ sudo apt-get install unzip
```

Finalmente, hemos de ejecutar el archivo .sh que acabamos de descomprimir. Para ello, desde consola, podemos movernos a la ruta donde se encuentre el fichero y escribir en terminal:

```
$ chmod +x Bateria_de_Ejecucion.sh
$ ./Bateria_de_Ejecucion.sh
```

Deberían comenzar las ejecuciones en Spark, una tras otra, hasta que finalicen todas.

### Lanzamiento en el servicio Google Cloud Dataproc

Google Cloud Dataproc es uno de los muchos servicios que ofrece la plataforma Google Cloud (https://cloud.google.com/). Para acceder a todos ellos es necesario contar con una cuenta Google, cuyo proceso de registro no será descrito en este manual. Igualmente, existen diferentes maneras de utilizar este servicio. En este apartado se describirá únicamente la manera de uso gráfica, por ser la más intuitiva para realizar labores sencillas.

Lo primero que debemos saber es que los servicios ofrecidos en Google Cloud (incluido el propio Google Cloud Dataproc) suelen están asociados a un proyecto concreto, aunque, si acabamos de registrarnos, ya contaremos con un proyecto creado. De querer generar uno nuevo podemos hacerlo en el menú superior, clic sobre el nombre del proyecto actual y clic sobre "Crear proyecto" en el menú desplegable que generará la acción anterior (ver imagen D.8).

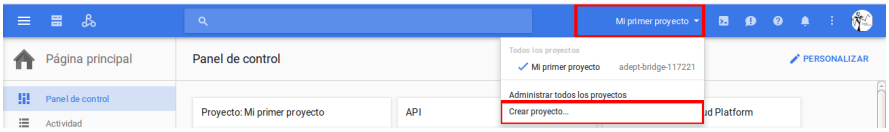


Figura D.8: Creación de un nuevo proyecto en Google Cloud.

Es importante indicar que, en el caso de estar realizando esta tarea con la versión de prueba de Google Cloud, no deberemos crear un nuevo proyecto, pues la condición gratuita se aplica solo al proyecto generado por defecto por Google.

Una vez tengamos nuestro proyecto nos encontraremos ante su página de inicio (ver imagen D.9), aunque no entraremos a explicar el funcionamiento de la misma.

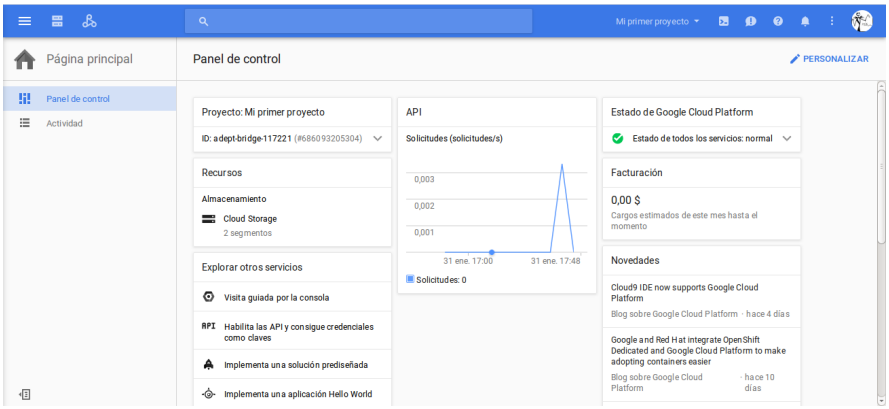


Figura D.9: Página de inicio de un proyecto en Google Cloud.

El segundo paso será almacenar nuestro archivo .jar, junto con los conjuntos de datos, en algún lugar que Google pueda alcanzar. Para ello, lo más sencillo será usar un segundo servicio de Google Cloud: Google Cloud Storage. Podemos acceder a él desde el menú superior, haciendo clic sobre el icono situado más a la izquierda y seleccionando el servicio "Storage" de entre las opciones que muestre el menú emergente (ver imagen D.10).

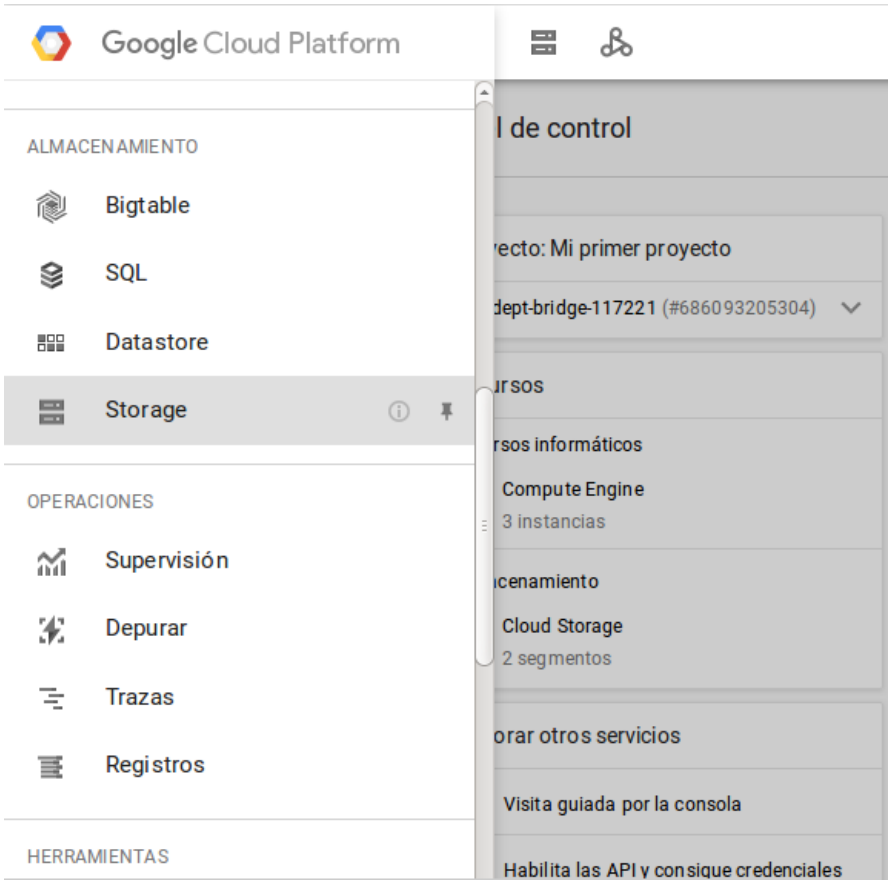


Figura D.10: Menú de selección de servicios de Google Cloud.

Si el proyecto es nuevo, es posible que tengamos que reservar un nuevo segmento de memoria para poder guardar nuestros datos. Simplemente hacemos clic sobre la opción "Crear segmento" que veremos en el centro de nuestra pantalla y rellenamos los campos de "Nombre", "Clase de almacenamiento" y "Ubicación" acorde a nuestras preferencias. Una vez terminado este paso, podemos acceder a nuestro nuevo espacio de memoria y administrarlo según nuestro gusto mediante las opciones del menú superior o arrastrando hacia la ventana del navegador todos aquellos elementos que deseamos que sean subidos al servicio de almacenamiento.

Ahora, necesitar asignar ciertos permisos a nuestro proyecto para poder asignarle un clúster. Para ello, vamos a visitar el servicio "Compute Engine" de la misma manera que antes visitamos "Storage", presionando el icono de la izquierda en el menú superior y buscando el servicio entre la lista de posibles. Lo primero que se nos preguntará es si deseamos asignar una serie de permisos a nuestro proyecto, a lo cual aceptaremos.

Con todo preparado, vamos a acceder finalmente al servicio que realmente nos interesa: Google Cloud Dataproc. Repetimos la misma operación anterior y buscamos servicio "Dataproc".

Ahora, vamos a crear nuestro primer clúster. Presionamos sobre el botón "Crear nueva agrupación" que veremos en el centro de la pantalla y rellenamos el formulario que permitirá crear un clúster. Es importante saber que:

- Es conveniente situar la ubicación del clúster en la misma que el segmento de memoria que está destinado a usar.
- Si estamos ejecutando desde la versión gratuita de Google Cloud, el tamaño del clúster estará limitado a 8 CPUs, incluyendo el nodo maestro.
- En el apartado "Opciones de acceso, inicialización, versión, red, segmento y trabajadores prioritarios" podemos seleccionar la "versión de imagen" del clúster, lo que definirá la versión de los productos que vamos a usar. Este proyecto se ha probado con la versión 0.2 que incluye Spark 1.5.2
- También en el apartado "Opciones de acceso, inicialización, versión, red, segmento y trabajadores prioritarios" hay una opción llamada "Segmento de aplicación de fases de Cloud Storage". Aquí debemos indicar el segmento de memoria que hemos creado anteriormente. Si no especificamos nada se creará uno por defecto.


Figura D.11: Pantalla principal de Google Cloud Dataproc

Bien, con nuestro clúster creado únicamente falta definir la tarea que debemos ejecutar. Para ello, en el menú lateral izquierdo podremos ver una sección llamada "Tareas" (ver imagen **??**). Nos podemos dirigir a ella y crear una nueva tarea del mismo modo que hemos creado anteriormente un segmento de memoria o una agrupación. De nuevo, es importante tener en cuenta que:

- Deberemos definir el "Tipo de tarea" como "Spark"

- La ruta al archivo .jar (o al fichero que contenga el conjunto de datos cuando introduzcamos las opciones) tendrá una estructura similar a "gs://nombreSegmento/directorio/fichero" si hemos utilizado el servicio Google Cloud Dataproc.

Podemos ver un ejemplo de una configuración en la imagen D.12.

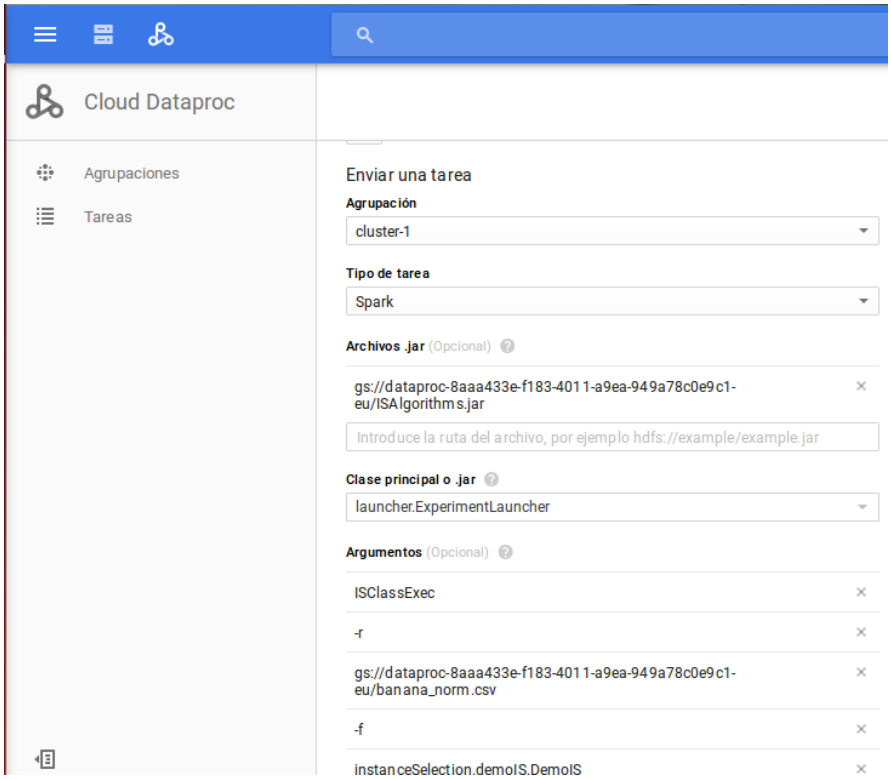


Figura D.12: Ejemplo de una tarea configurada

Lanzada la aplicación seremos redirigidos a una nueva pantalla donde podremos observar una línea de comandos tal y como si hubiésemos lanzado la aplicación en nuestra propia máquina. Mientras la ejecución tiene lugar podemos realizar cualquier otro tipo de operación, incluido definir nuevas tareas.

Una vez terminada la tarea que hayamos programado, queda ver el fichero resultante de la aplicación, para lo que deberemos conectarnos al nodo maestro. Accedemos a la vista principal de nuestro clúster desde la pantalla principal de Google Dataproc (ver imagen D.11) y allí hacemos click sobre "Instancias de VM" y sobre el botón "SSH" que aparecerá junto al nombre del nodo maestro. Esto abrirá una nueva ventana del navegador que emulará una consola de comandos. No es objetivo de esta sección hablar de todas las posi-

bles opciones que podemos realizar, únicamente veremos el fichero resultante ejecutando el siguiente comando:

```
$ cat /tmp/nombre_tarea/results/nombre_fichero_resultados
```

### Cheatsheet

A continuación se va realizar un listado con todas las posibles opciones que actualmente pueden formar parte de la sentencia de invocación del programa.

| Parámetro | Descripción |
|---|---|
| Tipo de ejecución | |
| ISClassExec | Define una tarea en la que intervenga un algoritmo de selección de instancias y un clasificador |
| ISClassExecTest | Define una tarea en la que intervenga un algoritmo de selección de instancias y un clasificador y, además mide el tiempo de ejecución de la labor de filtrado. Este último modo de lanzamiento podría afectar ligeramente a la ejecución de la tarea, aumentando su tiempo de ejecución. Para más información ver **??** |
| Parámetros del lector | |
| -f | Indica que el atributo de clase es el primero de los atributos de cada instancia. |
| -hl + Int | Indica que existe una cabecera en el fichero y cuantas líneas forman dicha cabecera. |
| Parámetros para el selector de instancias | |
| instanceSelection.lshis.LSHIS | |
| -and + Int | Número de funciones AND. |
| -or + Int | Número de funciones OR. |
| -w + Double | Anchura de los *buckets*. |
| -s + Long | Semilla utilizada para generar números aleatorios. |
| instanceSelection.demoIS.DemoIS | |
| -rep + Int | Número de votaciones a realizar. |
| -alpha + Double | Valor alpha. |
| -np + Int | Número de particiones en las que se dividirá el conjunto de datos original. |
| -dsperc + Double | Porcentaje del conjunto de datos utilizado para calcular el error durante el cómputo del fitness. |
| -s + Long | Semilla utilizada para generar números aleatorios. |
| Parámetros para el clasificador | |
| classification.seq.knn.KNN | |
| -k + Int | Número de vecinos más cercanos |

Cuadro D.1: Cheatsheet

# Bibliografía


[1] J. Alcalá Fdez, A. Fernandez, J. Luengo, J. Derrac, S García, L. Sánchez, and F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17:255–287, 2011.

[2] J. Alcalá Fdez, L. Sánchez, M.J García, S. del Jesus, S. Ventura, J.M Garrell, J. Otero, C. Romero, J. Bacardit, M.V Rivas, J.C. Fernández, and F. Herrera. KEEL: A Software Tool to Assess Evolutionary Algorithms to Data Mining Problems. *Soft Computing*, 13:307–318, 2009.

[3] Álvar Arnáiz González, José Francisco Díez Pastor, César García Osorio, and Juan José Rodríguez Díez. Herramienta de apoyo a la docencia de algoritmos de selección de instancias. In *Jornadas de Enseñanza de la Informática*. Universidad de Castilla-La Mancha, 2012.

[4] Álvar Arnaiz-González, José F. Diez Pastor, César García Osorio, and Juan J. Rodrígez. LSH-IS: Un nuevo algoritmo de selección de instancias de complejidad lineal para grandes conjuntos de datos. In *Actas de la XVI Conferencia de la Asociación Española para la Inteligencia Artificial CAEPIA 2015*. Asociación Española para la Inteligencia Artificial, 2015.

[5] Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. Integrating apache spark into pbs-based hpc environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 34. ACM, 2015.

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

[7] Donnie Berkholz. The emergence of spark, 2015. http://redmonk.com/dberkholz/2015/03/13/the-emergence-of-spark/.

[8] Louis Columbus. 84 % of enterprises see big data analytics changing their industries' competitive landscapes in the next year, 2014. www.forbes.com/sites/louiscolumbus/2014/10/19/84-of-enterprises-see-big-data-analytics-changing-their-industries-competiti#d25708c32502.

[9] César García-Osorio, Aida de Haro-García, and Nicolás García-Pedrajas. Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artificial Intelligence*, 174(5–6):410 – 441, 2010. http://www.sciencedirect.com/science/article/pii/S0004370210000123.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Petter Reutemann, and Ian H. The weka data mining software: An update. *SIGKDD Explorations*, 11, 2009.

[11] Fraunhofer IAIS. Fraunhofer Intelligent Data Analysis Group Benchmark Repository. http://www.iais.fraunhofer.de/index.php?id=32&L=1.

[12] Google Inc. Google Cloud Dataproc. https://cloud.google.com/dataproc/.

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

[14] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning Spark: Lightning-Fast Big Data Analysis*. .ºReilly Media, Inc.", 2015. chapter 7.

[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] Oracle. What are the system requirements for java? http://java.com/en/download/help/sysreq.xml.

[17] Spark. Machine Learning Library (MLlib) Guide - Dependencies, 2015. http://spark.apache.org/docs/latest/mllib-guide.html#dependencies.

[18] Apache Spark. Hardware Provisioning. https://spark.apache.org/docs/1.5.1/hardware-provisioning.html.

[19] Apache Spark. Spark Standalone Mode. http://spark.apache.org/docs/latest/spark-standalone.html.

[20] "WebUpd8" team. Oracle Java Installer. https://launchpad.net/~webupd8team/+archive/ubuntu/java.

[21] 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, visited 2015-10-28.

[22] The University of Waikato Weka. Can I use CSV files?, 2009. https://weka.wikispaces.com/Can+I+use+CSV+files%3F, visited 2015-10-28.