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 — 30 de enero 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 (kNN).

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 [10] 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 B.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 [11]. 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.

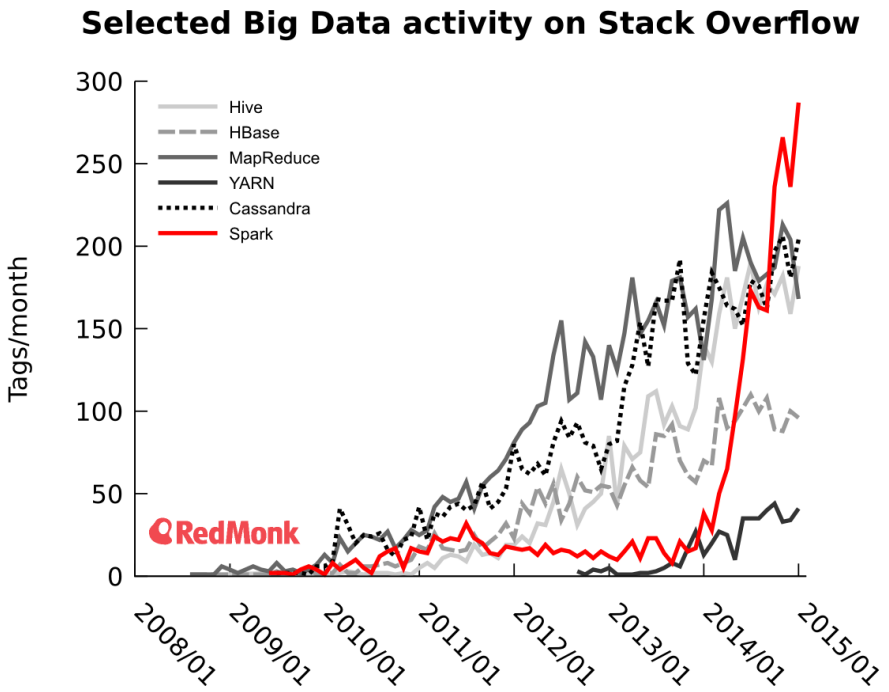

Figura A.1: Evolución del interés por Apache Spark en la plataforma Stack Overflow[6]

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 [7].

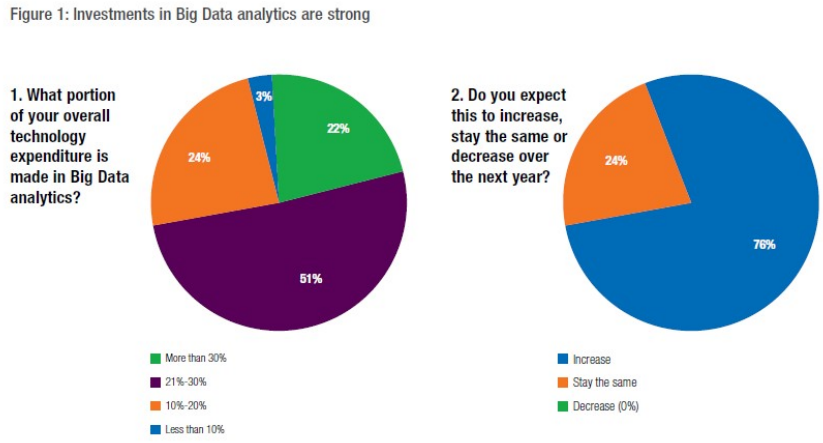

Figura A.2: Interés de las empresas por apostar por el Big Data (2014) [7]

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 [8] 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

Pruebas del sistema

B.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 [12]	2310	19	7
Banana shaped [9]	5300	2	2
Pen-Based Recognition of Handwritten Digits [12]	10992	16	10
Letter Recognition [12]	20000	16	26
Human Activity Recognition [14]	165.632	17	5
Covertype [12]	581.012	54	6
Poker [12]	1.025.010	10	10
HIGGS [12] [5]	11.000.000[1]	28	2

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

B.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 B.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.

B.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 [15].
- 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 B.4 y B.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*.

B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.2 o incluso en la tabla B.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 B.1

- Nótese en la tabla B.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 B.2.

- Por lo general, y a la vista de la gráfica B.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 B.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 B.5 o B.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 B.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.

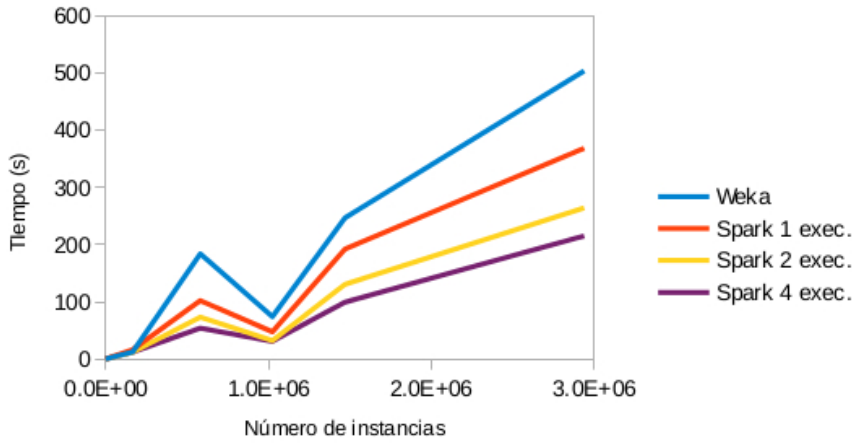

Figura B.1: Evolución del tiempo de clasificación según el número de instancias clasificadas.

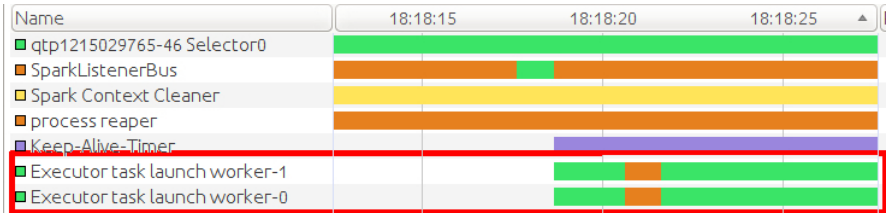

Figura B.2: Hilos de ejecución de Spark en el uso del algoritmo Naive Bayes sobre "Poker" con 4 procesadores asignados.

B.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 kNN 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 B.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* [13]. 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 B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.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 B.13. Durante el resto de mediciones hemos conseguido ejecuciones hasta un 50 % más rápidas con Spark (ver B.16)

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

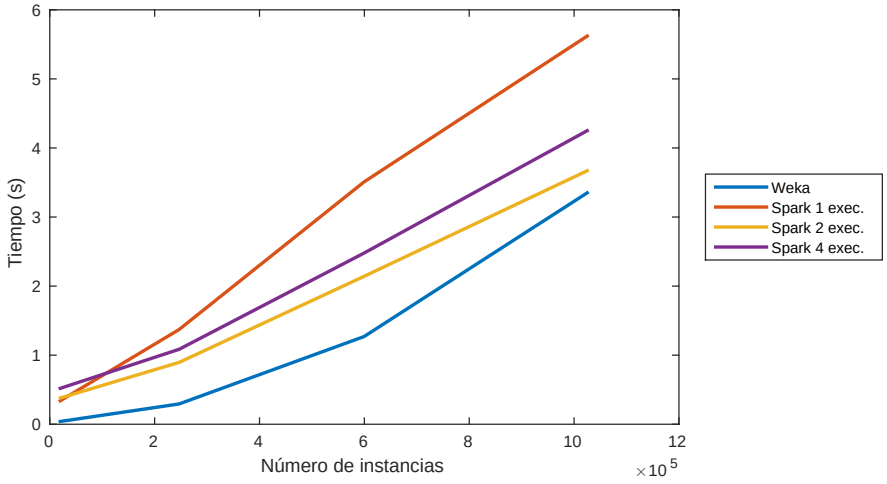

Figura B.3: Evolución del tiempo de ejecución según el tamaño del conjunto de datos usando LSHIS.

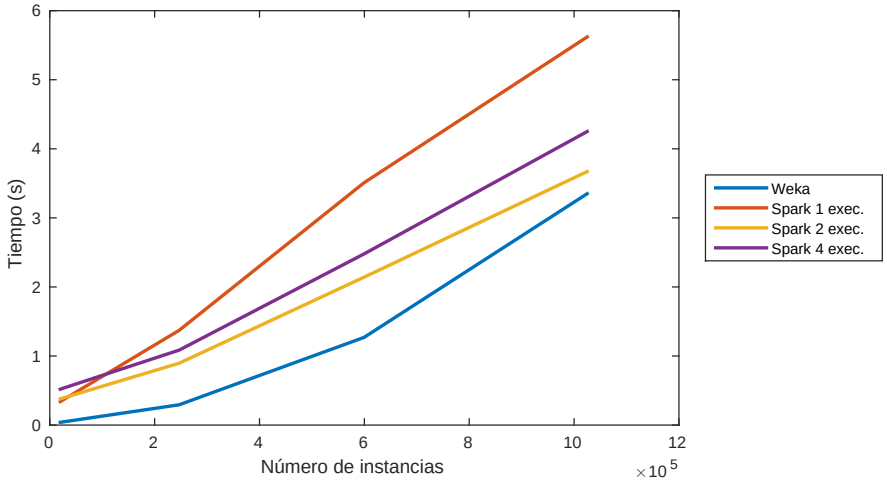

Figura B.4: Evolución del tiempo de ejecución según el tamaño del conjunto de datos usando DemoIS.

- 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 B.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 B.8 y tabla B.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 B.13 y tabla B.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.

B.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 B.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 [4].

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.

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] 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.

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

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

[7] 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-competitive-lands#d25708c32502.

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

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

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

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

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

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

[14] 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.

[15] 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.