Paralelización de algoritmos de selección de características en la plataforma Weka*

Carlos Eiras-Franco[1], Verónica Bolón-Canedo[1], Sabela Ramos[2], Jorge González-Domínguez[3], Amparo Alonso-Betanzos[1], and Juan Touriño[2]

[1] Computer Science Dept., University of A Coruña, 15071 A Coruña, Spain
carlos.eiras.franco@udc.es, veronica.bolon@udc.es, ciamparo@udc.es
[2] Dept. of Electronics and Systems, University of A Coruña, 15071 A Coruña, Spain
sabela.ramos.garea@udc.es, juan@udc.es
[3] Parallel and Distributed Architectures Group, Institute of Computer Science, Johannes Gutenberg University, Staudingerweg 9, 55128 Mainz, Germany
j.gonzalez@uni-mainz.de

Resumen Cada día se genera una abrumadora cantidad de datos, lo que se traduce en un volumen de información difícil de analizar. Técnicas como la selección de características se hacen imprescindibles a la hora de abordar conjuntos de datos grandes. Entre las herramientas que ofrecen esta funcionalidad, Weka es una de las más populares, aunque su rendimiento se resiente al procesar grandes conjuntos de datos, empleando tiempos demasiado largos para resultar prácticos. El procesamiento en paralelo puede ayudar a solventar este problema; en particular, la potencia computacional de las máquinas multi-núcleo se puede explotar utilizando procesamiento multi-hilo. Con esta técnica se puede conseguir una aceleración del proceso de selección de características en entornos de una sola máquina, permitiendo a los usuarios trabajar con conjuntos de datos mayores. En este trabajo se propone una versión multi-hilo de dos conocidos algoritmos de selección de características incluidos en Weka. Los resultados obtenidos en pruebas sobre cinco conjuntos de datos reales muestran que el uso de las nuevas versiones permite realizar la selección de características en hasta 16 veces menos tiempo.

1. Introducción

La dimensionalidad (muestras × características) de los conjuntos de datos analizados en problemas de aprendizaje máquina ha ido creciendo constantemente durante los últimos años. Tomando como referencia los conjuntos de datos publicados en la popular libSVM Database [1], el tamaño de dichos conjuntos se ha multiplicado por 500. Manejar estos volúmenes de datos constituye un reto para los algoritmos de aprendizaje máquina tradicionales, dado que el sobreajuste

* Este trabajo ha sido en parte financiado por la Xunta de Galicia bajo la Red de Investigación R2014/041 y el proyecto GRC2014/035, y por el Ministerio de Economía y Competitividad bajo el proyecto TIN2012-37954, parcialmente financiado con fondos FEDER de la Unión Europea.

puede tener un impacto negativo en su rendimiento, los modelos más complejos son más difíciles de interpretar y tanto la velocidad como la eficiencia de estos algoritmos decaen con el aumento de la dimensionalidad. Esta situación ha provocado la aparición de numerosas técnicas diseñadas para analizar conjuntos de alta dimensionalidad. Para los conjuntos con muchas variables, las técnicas de selección de características son imprescindibles. La selección de características consiste en determinar aquellas que son relevantes e intentar eliminar toda la información irrelevante o redundante que sea posible, sin que esto se traduzca en una degradación en el rendimiento del clasificador. La herramienta Weka (Waikato Environment for Knowledge Analysis) [2] es una plataforma de aprendizaje máquina muy popular que ha sido descargada más de seis millones de veces. Se puede utilizar como una aplicación autónoma o importarse como librería desde el código del usuario. Sin embargo, alguno de los algoritmos implementados en Weka tienen problemas para manejar conjuntos de datos grandes, demorándose en exceso. Una mejora en la eficiencia temporal de estos algoritmos permitirá a sus numerosos usuarios procesar grandes conjuntos de datos, hasta ahora fuera del alcance de esta herramienta.

Para encarar el problema de la alta dimensionalidad, durante la última década se han creado nuevas soluciones de programación paralela, como MapReduce [3], implementada en la plataforma de código abierto Apache Hadoop [4] o, más recientemente, Apache Spark [5], propuesta como una solución para el análisis de Big Data. La aparición de estas tecnologías llevó a la creación de librerías de aprendizaje máquina paralelas: Mahout [6] que se ejecuta sobre Apache Hadoop, y MLlib [7] que utiliza Apache Spark, son algunos ejemplos. Aunque estas librerías contienen una amplia variedad de algoritmos de aprendizaje máquina, no ofrecen muchas opciones en lo tocante a selección de características. El gran abanico de algoritmos que incluye Weka hacen que su uso sea generalizado entre los analistas de datos. Además, para sacar provecho de las librerías Mahout o MLlib, el usuario necesita tener una instalación de Hadoop o Spark que, aunque permiten su ejecución en una sola máquina, precisan de un cluster de ordenadores para ser verdaderamente ventajosas. Por el contrario, Weka se ejecuta sobre Java y está diseñado para ejecutarse en máquinas aisladas, lo que lo hace muy adecuado para el usuario medio. Sin embargo, la implementación de Weka de algunos algoritmos arroja tiempos de ejecución muy elevados, limitando en la práctica su uso a conjuntos pequeños de datos.

Utilizar procesamiento paralelo para realizar selección de características es un enfoque que se ha explorado en el pasado [8]. El objetivo de este trabajo es mostrar que se puede utilizar el procesamiento multi-hilo para obtener versiones más rápidas de los algoritmos de selección de características incluidos en Weka, permitiendo a los usuarios analizar mayores conjuntos de datos en menos tiempo.

2. Selección de características

Por selección de características se conoce al proceso que detecta características relevantes y descarta aquellas que son redundantes o irrelevantes. El objetivo

de esta técnica es obtener un subconjunto de características que ofrezca mínima degradación en el rendimiento al ser utilizado por un clasificador, al tiempo que describe el problema dado con exactitud. Simplifica el conjunto de datos tanto en tamaño como en dificultad de comprensión [9], lo que se traduce en algoritmos de clasificación más simples y veloces, mayor sencillez de interpretación del problema y menores requisitos de almacenamiento.

Los métodos de selección de características se pueden clasificar en dos categorías: evaluadores individuales o evaluadores de subconjuntos. Los individuales también se denominan rankers y asignan un peso a cada atributo que representa su relevancia. Los evaluadores de subconjuntos, por el contrario, se valen de una estrategia de búsqueda para determinar un subconjunto de características candidato y tienen la ventaja de que eliminan atributos redundantes, con el coste de que son más complejos. De acuerdo con la relación entre el método de aprendizaje utilizado, los métodos de selección de características pueden ser [10] **filtros**, que son métodos que se aplican independientemente del proceso inductivo y tienen, por lo general, bajo coste computacional; **wrappers**, que utilizan el algoritmo inductivo como caja negra para evaluar la validez de cada subconjunto candidato y que, como resultado, son algoritmos más costosos computacionalmente pero más precisos; o **métodos embebidos**, que realizan la selección de características en el proceso de entrenamiento y suelen ser específicos a ciertos algoritmos de aprendizaje.

Dado que los filtros son los menos exigentes computacionalmente de los tres tipos de métodos, son los más adecuados para tratar con conjuntos de alta dimensionalidad. En esta publicación se han seleccionado dos de los filtros más comúnmente utilizados para ser reimplementados con un enfoque paralelo: ReliefF e InfoGain. Ambos son rankers, puesto que devuelven una ordenación de las características, que serán descartadas por el usuario a partir de un umbral de su elección. Se incluyen en la suite Weka y se detallan en las Secciones 3.2 y 3.3, respectivamente.

3. Método de paralelización propuesto

Las reimplementaciones de ReliefF e InfoGain con procesamiento multi-hilo buscan obtener los mismos resultados en términos de atributos seleccionados pero reduciendo significativamente el tiempo empleado para facultar al usuario para procesar conjuntos más grandes en un tiempo razonable.

3.1. Procesamiento multi-hilo

Desde finales de los años 90 existen procesadores con la capacidad de ejecutar tareas independientes simultáneas, también conocidas como hilos [11]. Originalmente el procesamiento multi-hilo consistía en ejecutar un hilo mientras otro estaba bloqueado, pero nuevas generaciones de procesadores ofrecen potencia computacional adicional mediante el incremento del número de unidades de procesamiento, llamadas núcleos, que los capacita para ejecutar múltiples hilos

simultáneamente. Para beneficiarse de esta capacidad del procesador, las aplicaciones pueden crear y gestionar hilos para cada tarea independiente. Estos hilos se ejecutan en el mismo espacio de memoria, lo que posibilita que se comuniquen mediante memoria compartida, aunque esto requiere medidas de sincronización que garanticen la consistencia de los datos y eviten condiciones de carrera.

Aunque la creación y gestión de hilos conlleva un coste computacional, este es mucho menos significativo que el que requieren los procesos. Aún así, este coste añadido hace que el uso de hilos sea subóptimo cuando las tareas paralelizadas tienen baja complejidad.

Java soporta la programación paralela en el núcleo del lenguaje. Esta característica permite a los programadores escribir código multi-hilo sin necesidad de utilizar librerías externas. En nuestras implementaciones el trabajo se dividió en tantas tareas como hilos queríamos utilizar, y después se creó un hilo para cada tarea. Este enfoque evita la necesidad de un gestor de hilos que se encargue de controlar su ejecución.

3.2. Algoritmo ReliefF

ReliefF [12] es un estimador heurístico creado a partir del algoritmo Relief [13] que maneja eficientemente conjuntos de datos incompletos y con ruido, sean binarios o multiclase. Este algoritmo multivariado recorre un grupo de muestras R buscando para cada una sus k vecinos más cercanos de la misma clase, denominados aciertos más cercanos H_j, y los k vecinos más cercanos de cada una de las otras clases, denominados fallos más cercanos $M_j(C)$. Cuando se han encontrado todos los vecinos se actualiza el peso de cada atributo $W[A]$ restándole la media ponderada (calculada con la función $diff$, que devuelve la distancia Manhattan entre dos muestras) de cada acierto H_j y sumándole, para cada clase C distinta de la de R, la media ponderada de la distancia a cada fallo $M_j(C)$. Para computar las medias, las distancias son ponderadas por la probabilidad P de la clase y divididas por el número total de muestras m. Este proceso se detalla en el Algoritmo 1. El proceso de encontrar los vecinos más cercanos de cada muestra (descrito en la Línea 2) es muy largo puesto que requiere compararla con el resto de muestras. Esta búsqueda se puede realizar independientemente para cada muestra y, por tanto, puede efectuarse en paralelo sin problemas de sincronización.

3.3. Algoritmo InfoGain

El algoritmo InfoGain [14] es un filtro univariado que calcula la información mutua de las diferentes características con respecto a la clase para generar una ordenación de las mismas. Asigna el peso (W) de cada atributo (A) comparando su ganancia de información con respecto a la clase. Para calcular este valor, la entropía (H) de cada clase dado ese atributo se resta de la entropía de dicha clase: $InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute)$. La entropía de una variable se define como $-\sum_i p(i) * log(p(i))$, donde i recorre todos los posibles valores de la variable. La probabilidad observada de que la variable tome

Algoritmo 1: Pseudo-código de ReliefF multi-hilo

Input: R ←Grupo de muestras con A atributos y clasificadas en C clases
Output: W ← Peso de cada atributo
1 Establecer pesos **W[A]** ← 0,0
2 **for** $i \leftarrow$ *1* ***to*** *HILOS_DISPONIBLES* **do in parallel**
3 $S_i \leftarrow$ Subconjunto disjunto de muestras
4 **foreach** I ***in*** S_i **do**
5 Encontrar los k aciertos más cercanos H_j
6 **for** *cada clase* $C \neq clase(I)$ **do**
7 De la clase C encontrar los k fallos más cercanos $M_j(C)$
end
end
end
8 **for** $A \leftarrow$ *1* ***to*** a **do**
9 **foreach** R_i ***in*** R **do**
10 **W[A]** ←
$$\mathbf{W[A]} - \sum_{j=1}^{k} \frac{diff(A,R_i,H_j)}{m*k} + \sum_{C \neq clase(R)} \left[\frac{P(C)}{1-P(clase(R_i))} \sum_{j=1}^{k} \frac{diff(A,R_i,M_j(C))}{m*k} \right]$$
end
end

un valor dado se representa con $p(i)$, y se calcula como la relación entre casos en los que la variable toma dicho valor y el número total de apariciones de la variable. Weka implementa este cómputo recorriendo todo el conjunto de datos contando el número de apariciones de cada valor posible para cada característica, guardando los contadores en un array. Este array se utiliza después para computar la ganancia de información de cada atributo.

En la solución que proponemos, detallada en el Algoritmo 2, el recuento de cada valor posible se realiza en paralelo para un subconjunto de muestras (Línea 4). Esto requiere un paso adicional, descrito en la Línea 7, que combina los recuentos de cada hilo en un recuento general. Puesto que esta división se realiza sobre el número de muestras, será más efectiva cuando el conjunto de datos conste de muchas muestras. Para conjuntos de datos pequeños el paso acumulador adicional puede consumir más tiempo que el que se ahorra al contar en paralelo, pero para conjuntos de datos grandes el tiempo requerido para acumular los recuentos parciales es despreciable en comparación con el tiempo del proceso de recuento.

Por último, el proceso de obtención de los valores de ganancia de información a partir de los recuentos también se puede realizar de manera independiente para cada atributo, por tanto puede computarse en paralelo (Línea 11). Las funciones *Entropia* y *EntropiaCondicionada* mostradas en la Línea 14 representan el cálculo de $H(Clase)$ y $H(Clase|Atributo)$ respectivamente.

Algoritmo 2: Pseudo-código de InfoGain multi-hilo

```
   Input: R ←Grupo de muestras con A atributos y clasificadas en C clases
   Output: W ← Peso de cada atributo
 1 Establecer todos los recuentos ← 0,0
 2 for t ← 1 to HILOS_DISPONIBLES do in parallel
 3 |   S_t ← Subconjunto disjunto de muestras
 4 |   foreach I in S_t do
 5 |   |   foreach a in A do
 6 |   |   |   recuentos_{a,valor_I(a),clase(I)} ←
   |   |   |   recuentos_parciales_{t,a,valor_I(a),clase(I)} + peso(I)
   |   |   end
   |   end
   end
 7 for t ← 1 to HILOS_DISPONIBLES do
 8 |   foreach a in A do
 9 |   |   foreach v in valores_a do
10 |   |   |   foreach c in Clases do
   |   |   |   |   recuentos_{a,v,c} += recuentos_parciales_{t,a,v,c}
   |   |   |   end
   |   |   end
   |   end
   end
11 for t ← 1 to HILOS_DISPONIBLES do in parallel
12 |   A_t ← Subconjunto disjunto de atributos
13 |   foreach a in A_t do
14 |   |   W[a] ← Entropia(recuentos_a) − EntropiaCondicionada(recuentos_a)
   |   end
   end
```

4. Resultados experimentales

Dado que el objetivo de este trabajo es aprovechar el procesamiento multi-hilo para acelerar la selección de características, los atributos seleccionados y los pesos asignados por las nuevas versiones de los algoritmos son exactamente los mismos que los obtenidos por las versiones originales. Por tanto, estas nuevas versiones no suponen ninguna pérdida de precisión en la clasificación, sino que realizan la selección de características en un menor tiempo. Es por ello que los resultados presentados a continuación se refieren únicamente al tiempo de ejecución del proceso de selección de características.

4.1. Configuración experimental

Para obtener una gama variada de escenarios en los que probar las implementaciones propuestas se han seleccionado cinco conjuntos de datos reales de alta dimensionalidad (sus características se describen en la Tabla 1).

Cuadro 1. Descripción de los conjuntos de datos

Conjunto	Características	Muestras	Clases
ColonCancer	2,000	61	2
Leukemia	7,129	71	2
Higgs	28	11,000,000	2
KDD99	41	4,898,430	23
Epsilon	2,000	500,000	2

Todos los resultados se obtuvieron en un ordenador multi-núcleo con dos procesadores Intel Xeon ES-2660 Sandy Bridge-EP [15]. Cada procesador consta de ocho núcleos a 2.20GHz. Este sistema tiene hyperthreading, i.e., se pueden utilizar dos hilos por núcleo y, por tanto, hay 32 hilos disponibles en total. La versión de Weka utilizada fue la 3.7.12 ejecutándose sobre OpenJDK 1.7.0_55. El SO instalado en la máquina era Rocks 6.1, basado en CentOS 6.x. La medida *speed-up* fue la elegida para cuantificar el rendimiento. Se define como la relación entre el tiempo secuencial original y el multi-hilo.

Cuadro 2. Tiempos de ejecución

ReliefF			
Conjunto	Tiempo (hh:mm:ss)		Speed-up
	Secuencias	Multi-hilo	
ColonCancer	00:00:00.752	**00:00:00.678**	1.11
Leukemia	00:00:01.766	**00:00:01.203**	1.47
Higgs (4 %)	29:20:24.550	**01:52:41.020**	15.62
KDD99 (10 %)	44:30:10.460	**02:40:31.640**	16.63
Epsilon (10 %)	23:12:36.490	**01:41:42.000**	13.69
InfoGain			
Conjunto	Tiempo (hh:mm:ss)		Speed-up
	Secuencial	Multi-hilo	
ColonCancer	**00:00:00.492**	00:00:00.552	0.89
Leukemia	**00:00:00.762**	00:00:00.859	0.89
Higgs	00:03:20.470	**00:03:11.733**	1.05
KDD99	00:02:25.455	**00:02:19.594**	1.04
Epsilon	00:08:02.721	**00:07:04.345**	1.14

4.2. Análisis de la implementación de ReliefF

La ausencia de dependencias entre iteraciones del algoritmo ReliefF (lo que a menudo se indica diciendo que es "embarazosamente paralelo") se traduce en descensos significativos en el tiempo de ejecución. A pesar de esta mejora, la complejidad de ReliefF crece cuadráticamente con el número de muestras y linealmente con el número de características y esto provoca que el tiempo de

ejecución sea elevado cuando el número de muestras es muy alto. Sin embargo, nuestra implementación puede aprovechar máquinas que tengan un gran número de núcleos, disminuyendo el tiempo de computación.

Para poder comparar con la versión secuencial hemos utilizado versiones reducidas de los conjuntos de datos más grandes, dado que con los conjuntos completos los tiempos requeridos por la versión secuencial serían inabarcables. Para los conjuntos *Epsilon* y $KDD99$ tomamos el primer 10 % de las muestras. El conjunto *Higgs* requirió mayor recorte, tomándose el primer 4 %. Tanto el conjunto *ColonCancer* como el *Leukemia* se dejaron intactos puesto que contienen un número reducido de muestras.

La Tabla 2 muestra los tiempos de ejecución de ambas implementaciones. La versión multi-hilo se ejecutó usando los 32 hilos disponibles[4]. Existe mejora del rendimiento incluso en los conjuntos de datos pequeños que requieren tiempos de procesamiento cortos. En dichos casos (conjuntos *ColonCancer* y *Leukemia*) los speed-up son bajos dado que el tiempo dedicado a gestionar los hilos es significativo en comparación con el tiempo total del procesamiento paralelo. Cuando el conjunto analizado es grande, el tiempo consumido en la gestión de los hilos se vuelve irrelevante frente a la ganancia de tiempo obtenida al realizar los cómputos en paralelo. La versión multi-hilo del algoritmo procesó los conjuntos grandes hasta 16 veces más deprisa que la versión secuencial.

4.3. Análisis de la implementación de InfoGain

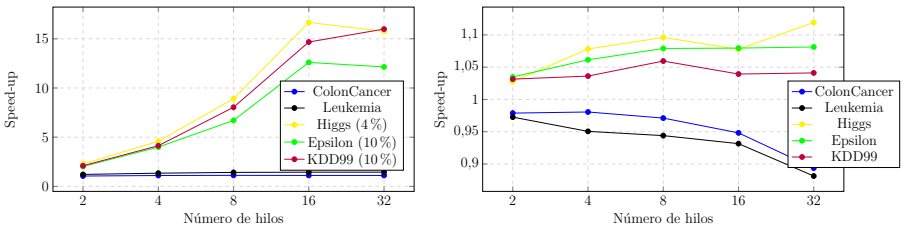
La implementación de Weka del algoritmo de selección de características InfoGain requiere que los atributos sean discretos, por lo que realiza un proceso de discretización antes de iniciar el proceso de selección de características. Esta discretización es independiente del algoritmo InfoGain, así que, para eliminar su impacto en el tiempo de ejecución y obtener una comparación más precisa de las dos versiones del algoritmo, todos los conjuntos de datos utilizados para probar InfoGain fueron discretizados previamente utilizando el mismo algoritmo que utiliza Weka [16]. La Tabla 2 muestra la comparativa entre los tiempos de ejecución de ambas versiones del algoritmo cuando se aplicó a distintos conjuntos. La versión multi-hilo se ejecutó empleando los 32 hilos disponibles.

El speed-up de la versión multi-hilo se hace perceptible cuando el tiempo de ejecución es más largo. Para procesos muy cortos, el sobrecoste asociado a la gestión de los hilos ensombrece la ganancia de tiempo en el proceso de selección, pero para trabajos más largos hay una ligera mejora. No obstante, un análisis más pormenorizado de la implementación revela que la mayor parte del tiempo necesario para selección de características con InfoGain en Weka se dedica a preparar el conjunto de datos, primero leyéndolo de disco y luego comprobando que los atributos son adecuados para el algoritmo. El proceso de selección de características en sí mismo se realiza en un tiempo corto en comparación con el tiempo total de ejecución, de manera que incluso una mejora drástica en la eficiencia temporal del algoritmo conlleva un speed-up modesto del tiempo total.

[4] El mejor tiempo de cada conjunto se ha resaltado en negrita en todas las tablas.

4.4. Impacto del número de hilos en el rendimiento

Hemos realizado pruebas con distinto número de hilos procesando los mismos conjuntos de datos para ilustrar la relación entre el rendimiento y el número de hilos empleado. Estos resultados se muestran en la Figura 1.



(a) ReliefF (b) InfoGain

Figura 1. Speed-up vs Número de hilos

Para el algoritmo ReliefF la complejidad del cálculo necesario para realizar la selección de características es suficiente para que el uso de múltiples hilos resulte beneficioso incluso para conjuntos pequeños. Además, el algoritmo es escalable y el rendimiento es mayor a medida que aumentamos el número de núcleos utilizados.

En el caso del algoritmo InfoGain el tiempo requerido para realizar la selección de características es casi despreciable en comparación con el tiempo de ejecución total y, por ello, cualquier mejora derivada del uso de múltiples hilos queda anulada por el coste de gestionar dichos hilos. Este fenómeno se puede apreciar en el speed-up del algoritmo InfoGain con los conjuntos *ColonCancer* y *Leukemia*, que disminuye a medida que aumenta el número de hilos.

Para los conjuntos más grandes, el tiempo dedicado a selección de características es mayor y esto hace el uso de múltiples hilos aconsejable, tal como se muestra en el incremento del speed-up del algoritmo InfoGain al procesar el conjunto *Higgs* cuando se incrementa el número de hilos.

En nuestro experimento las máquinas que utilizamos para ejecutar los tests disponen de 16 núcleos independientes, cada uno capaz de ejecutar dos hilos. Cuando se utilizan 16 hilos, estos se dirigen a núcleos distintos con uso exclusivo de recursos, obteniendo el mayor rendimiento. Por el contrario, al solicitar 32 hilos, se sitúan 2 en cada núcleo, compitiendo así por los recursos del mismo que no están duplicados [15]. Esto tiene como consecuencia una degradación del rendimiento que, en nuestro mejor caso, apenas mejora el uso de 16 hilos.

5. Conclusiones

Este trabajo ha explorado nuevas implementaciones de dos populares algoritmos de selección de características incluidos en la suite de aprendizaje máquina

Weka. Hemos propuesto nuevas versiones que aprovechan el procesamiento multihilo para acelerar el cálculo, facultando a los usuarios para trabajar con mayores conjuntos de datos en un tiempo razonable.

Los resultados experimentales obtenidos muestran una mejora significativa en el tiempo de ejecución del algoritmo ReliefF, obteniendo speed-ups de más de 16 para conjuntos grandes de datos reales en una máquina multi-núcleo con 16 núcleos. Los mismos conjuntos se utilizaron para probar una nueva versión de InfoGain, arrojando una mejora respecto a la versión secuencial, aunque su corto tiempo de ejecución total hace la ganancia menos relevante. Como trabajo futuro planeamos reimplementar otros algoritmos de selección de características populares incluidos en Weka.

Referencias

1. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.
2. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
3. Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
4. Apache Hadoop Project. `http://hadoop.apache.org/`. Accessed: 2015-05-01.
5. Apache Spark: Lightning-fast cluster computing. `https://spark.apache.org/`. Accessed: 2015-05-01.
6. Apache Mahout Project. `http://mahout.apache.org/`. Accessed: 2015-05-01.
7. Machine Learning Library (MLlib) Guide. `http://spark.apache.org/docs/latest/mllib-guide.html`. Accessed: 2015-05-01.
8. Zheng Zhao, Ruiwen Zhang, James Cox, David Duling, and Warren Sarle. Massively parallel feature selection: an approach based on variance preservation. *Machine learning*, 92(1):195–220, 2013.
9. Verónica Bolón Canedo. *Novel feature selection methods for high dimensional data*. PhD thesis, 2014.
10. Isabelle Guyon. *Feature extraction: foundations and applications*, volume 207. Springer Science & Business Media, 2006.
11. Intel Hyper-Threading Technology - Technical Users Guide. `http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf/`. Accessed: 2015-05-01.
12. Igor Kononenko. Estimating attributes: analysis and extensions of RELIEF. In *Machine Learning: ECML-94*, pages 171–182. Springer, 1994.
13. Kenji Kira and Larry A Rendell. A practical approach to feature selection. In *Proceedings of the ninth international workshop on Machine learning*, pages 249–256, 1992.
14. J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
15. Subhash Saini, Johnny Chang, and Haoqiang Jin. Performance Evaluation of the Intel Sandy Bridge Based NASA Pleiades Using Scientific and Engineering Applications. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, pages 25–51. Springer, 2014.
16. Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029, 1993.