ANÁLISIS DE MINERÍA DE DATOS DISTRIBUIDA CON WEKA PARALLEL EN COMPUTADORAS CON MÚLTIPLES PROCESADORES FÍSICOS Y LÓGICOS

ISSN 2219-6722
ISSNE 2222-2707

Jairo Jonathan Martínez
Universidad de Costa Rica, San José, Costa Rica

RESUMEN

Los estudiantes e investigadores cada vez más tienen la necesidad de aplicar de minería de datos. Weka, surge como un proyecto que pone a su disposición una colección de algoritmos que facilitan el descubrimiento de conocimiento en los datos. Sin embargo, no fue diseñado para aprovechar el paralelismo local con multihilo ni los ambientes distribuidos. Ante esta necesidad han surgido algunos proyectos que extienden Weka permitiéndole aprovechar mejor los recursos. Uno de estos es Weka Parallel. En este artículo se analiza el rendimiento de Weka Parallel y la mejor forma de utilizar en una sola computadora que tenga procesador multinúcleo.

Palabras Clave: Minería de Datos, weka, weka parallel, programación distribuida

ANALYSIS OF DATA MINING DISTRIBUTED WITH WEKA PARALLEL IN COMPUTERS WITH MULTIPLE PHYSICAL AND LOGICAL PROCESSORS

ISSN 2219-6722
ISSNE 2222-2707

Jairo Jonathan Martínez
Universidad de Costa Rica, San José, Costa Rica

ABSTRACT

Students and researchers increasingly have the need for data mining. Weka, emerges as a project that offers a collection of algorithms that facilitate knowledge discovery in data. However, it was not designed to take local multithreaded parallel or distributed environments. Given this need, some projects that extend Weka allowing better use of resources, have emerged. One of these is Weka Parallel. This article analyzes Weka Parallel performance and the best way to use a single computer that has a multicore processor for distributed data mining.

Key words: data mining, weka, weka parallel, distributed programming

1. INTRODUCCIÓN

Una variedad de ciencias e industrias están generando grandes cantidades de datos provenientes de sensores y otras fuentes. En la biología se utilizan cientos de terabytes para guardar información del ADN. Las grandes cadenas comerciales y financieras juntan en su almacén de datos millones de transacciones [7]. La minería de datos provee de algoritmos necesarios para descubrir patrones y generar conocimiento automáticamente como parte del Descubrimiento de Conocimiento a partir de los Datos (DCD o KDD de sus siglas en inglés).

Los problemas que comúnmente se resuelven son detección de anomalías, aprendizaje de reglas de asociación, agrupamiento, clasificación, regresión y sumarización. Tienen su aplicación en un variado rango de dominios que van desde análisis de mercados, detección de fraudes, bioinformática, detección de correo no deseado y otros más [2]. Sin embargo, los algoritmos de minería de datos requieren gran poder computacional y toman mucho tiempo. Actualmente se trata de aprovechar la computación en la nube y otras estrategias para mejorar los tiempos de respuesta en el procesamiento de minería sobre grandes conjuntos de datos. Se han propuesto varias implementaciones como HC-CART, una implementación en paralelo del algoritmo CART[2], OpenPlanet una implementación de código abierto de PLANET, una algoritmo de Google basado en MapReduce [9]. GridWeka, Weka Parallel, Weka4WS y WekaG que son aplicaciones que extiende Weka para realizar minería de datos en grids[3]. Adicionalmente se han implementado algunas versiones que aprovechan el procesamiento en las GPU, como GPUMiner, un sistema de minería de datos en paralelo que utiliza los procesadores de video GPU (Graphics Processing Unit) [4] y una implementación más que se basa en Weka tratando de mejorar el tiempo de respuesta para el usuario final con una sola computadora usando el GPU en [3]. Adicionalmente las computadoras personales usadas por estudiantes e investigadores normalmente tienen varios núcleos físicos y/o lógicos. Para mejorar el tiempo de respuesta en minería de datos en computadoras personales multinúcleos y utilizar de la mejor manera las capacidades de hardware que se tienen, en este artículo se estudia Weka Parallel y se analiza el tiempo de respuesta para varios tamaños de conjuntos de datos, añadiendo servidores de ejecución en la misma computadora hasta alcanzar el número máximo de núcleos lógicos disponibles. En las siguientes secciones se introduce la minería de datos, luego se presenta Weka y Weka Parallel, para finalmente presentar el experimento, sus resultados y las conclusiones a las que se ha llegado y se menciona el trabajo que puede continuar realizándose en esta línea de investigación.

2. MARCO CONCEPTUAL

El DCD incluye además de la minería de datos la (1) preparación de datos, (2) la selec-

ción de datos, (3) la limpieza de los datos, (4) la incorporación del conocimiento previo adecuado y (5) la interpretación de los resultados. La minería de datos es la aplicación de algoritmos específicos para la extración de patrones. Es un campo multidiciplinario que se basa en técnicas de aprendizaje automático, reconocimiento de patrones y la estadística [5]. La minería de datos es un proceso complejo que aplicado sobre grandes volúmenes de datos toma mucho tiempo en ejecutarse. Las tareas que involucra se agrupan en el preprocesado del conjunto de datos, la generación del modelo de clasificación, la validación del modelo y la presentación de resultados [8]. La fase de validación del modelo realizada con cross validation es altamente paralelizable y por lo tanto se analiza en los párrafos siguientes, sin embargo, se presenta primero la forma en que se realiza la inducción de reglas para la clasificación.

Figura No. 1. Ejemplo de validación cruzada para 5 folds.

Clasificación

Para la inducción de reglas supervisada, se le otorga al algoritmo dos conjunto de datos previamente clasificados. Uno de ellos se utiliza para la generación del modelo y es conocido como el conjunto de entrenamiento. El otro se utiliza probar el modelo, a éste se le aplican las reglas encontradas y se compara el resultado de clasificación obtenido por el modelo con la clase que le fue asignada en el conjunto de datos.

Cross-Validation

Se realiza después de haber generado el modelo de minería de datos para determinar su validez y se considera la principal metodología para medir el éxito de los algoritmos de aprendizaje de máquina para describir los datos futuros. El método comúnmente utilizado es el N-folds cross-validation [1] que consiste en separar el conjunto de datos de entrenamiento en N subconjuntos (S) de tamaño aproximadamente igual y se repite el algoritmo de minería N veces. Para la iteración n se toma el subconjunto Sn como el

conjunto de prueba y los restantes N −1 subconjuntos se utilizan para el entrenamiento del modelo. La Figura 1 muestra gráficamente un ejemplo del proceso de validación cruzada para un conjunto de datos con N = 5 folds. Cada línea en la figura representa una ejecución del algoritmo y en diferentes colores se identifica los subconjuntos de datos utilizados para el entrenamiento y la prueba del modelo de esa iteración. Así, para la iteración el subconjunto S1 se convierte en el conjunto de prueba y el subconjunto se utiliza como el conjunto de entrenamiento para la generación del modelo. Desde luego, la validación cruzada con N-folds provee una seguridad estadística alta, pero requiere la ejecución del algoritmo N veces sobre el mismo conjunto de datos. Esto aumenta la necesidad computacional y por lo tanto multiplica el tiempo de espera para el usuario en al menos N veces, en un ambiente no paralelizado ni distribuido.

Minería de datos distribuida

La computación distribuida La minería de datos distribuida se utiliza para mejorar los tiempos de respuesta para el usuario y para lograrlo se han desarrollado nuevos algoritmos distribuidos implementados a partir de las versiones centralizadas [8]. Una implementación popular en el ambiente académico y organizacional para minería de datos es Weka. Sin embargo, el proyecto original de Weka no fue diseñado para computación distribuida por lo que ha sido extendida por varios proyectos para computación distribuida.

WEKA

El Waikato Environment for Knowledge Analysis (WEKA) es un proyecto desarrollado en la Universidad de Waikato de Nueva Zelanda. Weka provee una colección de algoritmos para el pre-procesamiento (filtrado) de datos y de minería de datos, incluyendo algoritmos de regresión, clasificación, agrupamiento, reglas de asociación y selección de atributos [6].

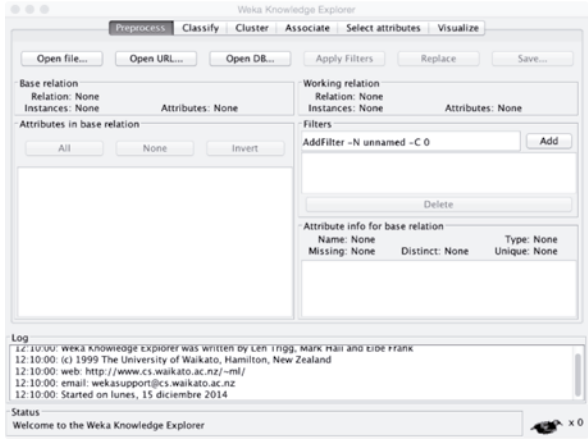


Figura No. 2. Interfaz de usuario del Explorer de Weka

Entornos de trabajo
Las interfaces de usuario de Weka son:
Simple CLI: Entorno consola para invocar directamente con java a los paquetes de weka.

Explorer: Entorno visual que ofrece una interfaz gráfica para el uso de los paquetes.

Experimenter: Entorno centrado en la automatización de tareas de manera que se facilite la realización de experimentos a gran escala.

KnowledgeFlow: Permite generar proyectos de minera de datos mediante la generación de flujos de información.

La Figura 2 muestra la interfaz de usuario para el Explorer de Weka. En la pestaña inicial se escoge la fuente de datos, que puede ser una base de datos, un texto plano con formato ARFF, o directamente de una URL (Localizador Universal de Recursos, de las siglas en inglés). La segunda pestaña da acceso a los algoritmos de clasificación y regresión. Esta interfaz se mantiene, con muy pocas variaciones en los proyectos que extiende Weka para sistemas distribuidos.

Modificaciones para Weka en Grids
El proyecto original de Weka no fue diseñado para aprovechar los procesadores con múltiples núcleos y menos para funcionar en diferentes computadoras. Para aliviar este problema se han creado proyectos que extienden Weka para adaptarlos a computación en grids. Entre estos se destacan Weka4WS, que utiliza web services, WekaG, GridWeka y Weka Parallel [3]. Este artículo se enfoca en WekaParallel y evalúa su rendimiento en una sola computadora multinúcleo.

Weka Parallel
La validación cruzada es paralelizable y representa una oportunidad para mejorar el rendimiento de la ejecución de los algoritmos. Weka Parallel (WP) es una modificación a Weka que implementa la ejecución en paralelo de la validación cruzada. Se sabe que cada *fold* en la validación cruzada puede ser calculado de forma independiente, WP modifica weka permitiendo que cada *fold* se ejecute en una computadora diferente.

Cada computadora disponible (servidor) abre una conexión y espera peticiones de ejecución en un puerto específico. Una vez iniciado los servidores, la computadora cliente puede conectarse a estos servidores para asignarles el trabajo. Cuando los servidores terminan los cálculos asignados los resultados se retornan al cliente [1]. En cada computadora puede ejecutarse más de un hilo de servidor, aprovechando los procesadores multinúcleos.

Configuración
La configuración es sencilla. Solo requiere crear un archivo en /.weka-parallel en sistemas UNIX/Linux y en c:\Windows\.weka-parallel para Windows. En este archivo la primera línea se debe indicar el número de puerto donde los servidores están escuchando, así: PORT =XXXX Las líneas siguientes llevan el nombre de cada uno de los servidores disponibles, uno por línea.

3. METODOLOGÍA

La investigación es de carácter cuantitativo y cuasi experimental. Se determina como cuasi experimental porque la selección de las unidades experimentales no pudo ser aleatoria por la naturaleza propia de las mismas. Variable de respuesta. Para el experimento se mide como variable de respuesta el tiempo de ejecución del algoritmo de minería de datos sobre cada tratamiento. Para medirlo se utiliza el comando time de la terminal de Unix. La herramienta ofrece tres diferentes tiempos: el tiempo real, el tiempo de usuario y el tiempo de sistema. El tiempo de usuario es el tiempo que realmente el proceso tiene acceso a los recursos del procesador, por lo tanto se utiliza el tiempo de usuario como variable de respuesta.

Factores de diseño
Se realizan dos etapas, primero se verifica el efecto de la cantidad de folds sobre el tiempo de ejecución y luego el efecto de la paralización.
Utilizan 3 factores:

- Tamaño de los conjuntos de datos: grandes (más de 1 MB) y pequeños (menos de 1MB).
- Cantidad de host: desde uno (sin paralizar) hasta ocho (cantidad de núcleos lógicos de la computadora de prueba).
- Número de folds: 5, 10 15 y 20 folds.

Factores contantes
Durante la ejecución de la investigación se utiliza varios factores que permanece constantes para asegurarse que el efecto sobre la variable de respuesta se deba solo a los factores de diseño. Los factores considerados constantes son:

- El sistema operativo afecta el tiempo de respuesta de los procesos por la forma en que asigna los recursos. Además, las llamadas al sistema se ejecutan de forma diferente en cada sistema. Para el experimento se mantuvo constante el sistema operático con Mac OSX 10.10.4.
- Hardware de la computadora. Se utiliza una computadora con procesador i7 de 4 núcleos físicos y 8 Lógicos. Tiene disco duro de estado sólido y una memoria RAM de 8GB.
- Algoritmo de minería. Cada algoritmo de minería de datos tiene diferentes necesidades computacionales. Para este experimento se utiliza el algoritmo de

clasificación J48, incorporado de manera natural en las distribuciones de Weka.
- Versión de Java. La versión de la máquina virtual de java sobre la que se ejecuta Weka es 1.8.0_25
- Se utiliza la versión 3.2.3 de Weka Parallel.

Factores de molestia

Los factores de ruido que no pueden controlarse durante la ejecución del experimento son:

- Asignación interna de los recursos del computador. El algoritmo de asignación de recursos de la computadora es no controlable, y además depende de factores que no están al alcance del experimentador.
- Servicios no controlados que se ejecutan en la computadora. Estos servicios se ejecutan el background. Estos afectan la forma en que la asignación de recursos es hecha y la paginación de la memoria RAM.
- Temperatura del procesador. La temperatura del procesador afecta la velocidad de los ciclos del reloj que controla el procesador, conocido como turboboost.

Réplicas

Cada unidad experimental se somete a la configuración de los diferentes tratamientos y se toman 10 réplicas para cada tratamiento. Se usaron dieciocho conjuntos de datos, bajo las condiciones descritas anteriormente se tomaron un poco más de cinco mil quinientas observaciones de tiempo de respuesta que se resumen en la sección de resultados.

Muestra de unidades experimentales

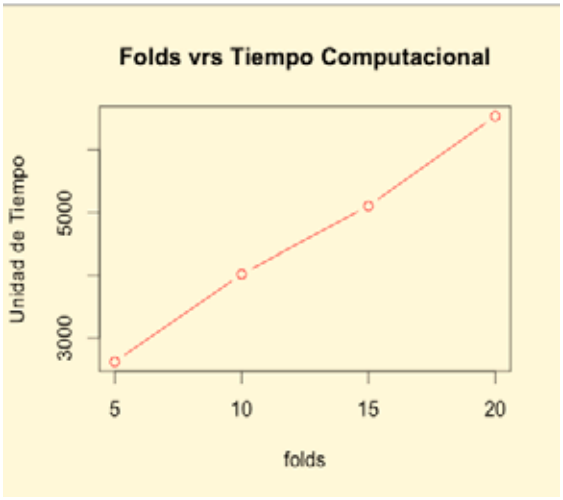
La muestra es no aleatoria. Se utilizaron conjuntos de datos en formato arff que están disponibles de forma gratuita en repositorios de la web. Los diferentes conjuntos se dividieron en dos grupos denominados como conjuntos grandes y conjuntos pequeños. Se consideraron como conjuntos grandes los archivos con más de 1MB de datos. Los conjuntos pequeños tienen menos de 1MB. En la Figura 3 se presenta un resumen descriptivo de la muestra de unidades experimentales.

Variable	Tamaño	Mean	Minimum	Q1	Median	Q3	Maximum
size	Grande	5182249	1012920	1266158	2944948	11335641	13826179
	Pequeño	89019	2890	13564	60374	180338	202935

Figura No. 3. Descriptivos de la muestra de unidades experimentales.

4. RESULTADOS

En esta sección se presentan los resultados obtenidos y un breve análisis de sus implicaciones.




```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.2243206  0.6893222  -7.579  0.01697 *
y            0.0038769  0.0001438  26.958  0.00137 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4142 on 2 degrees of freedom
Multiple R-squared:  0.9973,    Adjusted R-squared:  0.9959
F-statistic: 726.7 on 1 and 2 DF,  p-value: 0.001373
```

Figura No. 4. Relación entre el número de folds y el tiempo requerido para la ejecución.

Influencia de la cantidad de *folds* en el tiempo de ejecución.

El primer resultado relevante, como era de esperarse, el tiempo de respuesta es directamente proporcional a la cantidad de *folds*. La Figura 4 muestra la relación entre la cantidad de folds y el tiempo que se requiere para ejecutar el *cross validation* y el análisis de la regresión lineal entre las dos variables. Existe una fuerte correlación entre las variables, con un R – cuadrado = 0.9973. La Figura 5 se presenta también un gráfico de cajas para el tiempo de repuesta en función de la cantidad de *folds* utilizada.

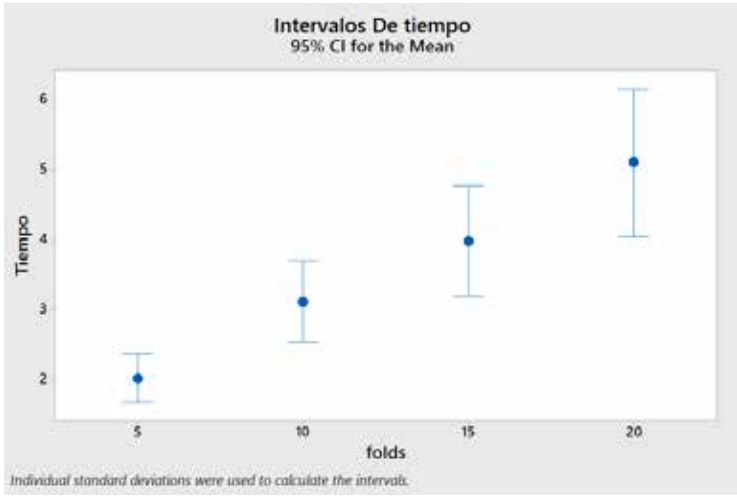


Figura No. 5. Grafica de cajas para el tiempo de respuesta en función de la cantidad de folds utilizados.

Tiempo de respuesta en función de los servidores disponibles.
En [1] se propone una diminución exponencial en el tiempo de respuesta utilizando catorce computadoras dual core. Una curva similar se encontró durante el experimento para conjuntos de datos grandes. En la Figura 6 se muestra una gráfica del tiempo de respuesta encontrado para un conjunto de datos grande. Se comenzó utilizando un solo servidor de ejecución, equivalente a no utilizar paralelismo. Luego se agregaron acumuladamente nuevos servidores en la misma computadora, tratando de optimizar el uso de los núcleos disponibles.

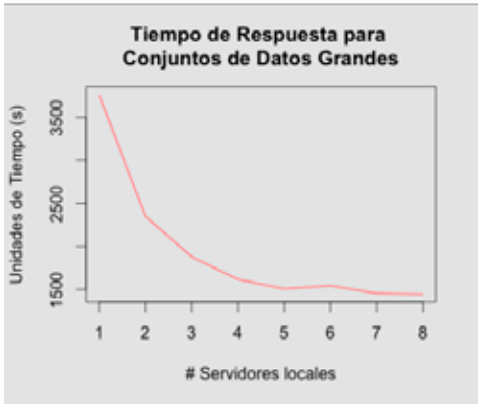


Figura No. 6. Tiempo de respuesta con varios núcleos para conjuntos de datos grandes.

Se nota una similitud con la curva propuesta en [1], sin embargo, se nota un decaimiento rápido luego de abrir más servidores que núcleos físicos en la computadora.

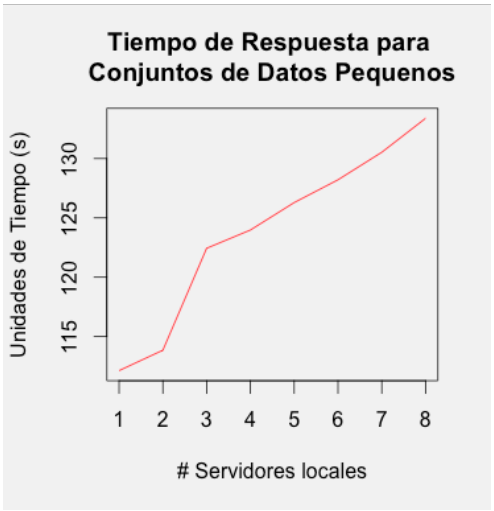


Figura No. 7. Tiempo de respuesta con varios núcleos para conjuntos de datos pequeños.

Al analizar conjuntos de datos más pequeños se obtiene un fenómeno diferente. Contrario a disminuir el tiempo, el *overhead* introducido por el paralelismo tiene como resultado un aumento en los tiempos de respuesta, puede verlo en la Figura 7.

5. TRABAJO FUTURO

Se puede tratar de encontrar el tamaño mínimo del conjunto de datos donde la paralelización con WP es beneficioso. También podría verificarse si esto depende solo del tamaño o también se ve afectado por otras variables, como la cantidad de campos y/o registros del conjunto de datos. Puede compararse los resultados obtenidos con WP repitiendo el mismo experimento con GridWeka o cualquier otro proyecto derivado.

6. CONCLUSIONES

Al aumentar la cantidad de *folds* se esperan por lo tanto un aumento proporcional en la cantidad de procesamiento necesario. El investigador puede decidir aumentar la cantidad de servidores locales de WP para mantener estable el tiempo de respuesta final. Sin embargo, en conjuntos de datos pequeños el tiempo no mejora, por lo que no se recomienda usar paralelismo. Un conjunto de datos grandes con tiempos de ejecución mayores al *overhead* introducido por el paralelismo, se recomienda iniciar servidores locales de WP tantos como núcleos físicos estén disponibles.

REFERENCIAS BIBLIOGRAFICAS

Celis, S., and Musicant, D. R. Weka-parallel: machine learning in parallel. In Carleton College, CS TR, Citeseer (2002).

Chrysos, G., Dagritzikos, P., Papaefstathiou, I., and Dollas, A. Hc-cart: A parallel system implementation of data mining classification and regression tree (cart) algorithm on a multi-fpga system. ACM Transactions on Architecture and Code Optimization (TACO) 9, 4 (2013), 47.

Engel, T. A., Chara˜o, A. S., Kirsch-Pinheiro, M., and Steffenel, L.-A. Performance improvement of data mining in weka through gpu acceleration. Procedia Computer Science 32 (2014), 93–100.

Fang, W., Lau, K. K., Lu, M., Xiao, X., Lam, C. K., Yang, P. Y., He, B., Luo, Q., Sander, P. V., and Yang, K. Parallel data mining on graphics processors. Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS08-07 2 (2008).

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. Advances in knowledge discovery and data mining.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: an update. ACM SIGKDD explorations newsletter 11, 1 (2009), 10–18.

Kumar, A., Kantardzic, M., and Madden, S. Distributed data mining.
Sánchez, A., Pena, J. M., Pérez, M. S., Robles, V., and Herrero, P. Improving distributed data mining techniques by means of a grid infrastructure. In On the Move to MeaningfulInternetSystems2004:OTM2004 Workshops, Springer (2004), 111–122.

Yin, W., Simmhan, Y., and Prasanna, V. K. Scalable regression tree learning on hadoop using openplanet. In Proceedings of third international workshop on MapReduce and its Applications Date, ACM (2012), 57–64.