Big Data y Algoritmos Genéticos Distribuidos en el aprendizaje de reglas de clasificación

Miguel Ángel Rodríguez[1,*], Antonio Peregrín[1]

[1] Universidad de Huelva, España
{miguel.rodriguez, peregrin}@dti.uhu.es

Resumen. Los algoritmos genéticos distribuidos son una potente herramienta de aprendizaje de reglas de clasificación que permite escalar eficientemente el tratamiento de grandes conjuntos de datos. Por otro lado el paradigma MapReduce permite tratar masivamente datos de manera distribuida de un modo simple. Este trabajo revisa brevemente las aportaciones realizadas y propone la implementación de MapReduce en un algoritmo distribuido para explotar las posibles ventajas de ambos modelos en un enfoque mixto que permitiría aprovechar las capacidades de aprendizaje distribuido del algoritmo genético con la mejora de rendimiento que permite MapReduce.

Palabras clave: Algoritmos Genéticos Distribuidos, MapReduce, Clasificación, Programación Distribuida, Programación Paralela.

1 Introducción

La revolución digital ha posibilitado que la captura de datos sea fácil y su almacenamiento tenga un coste muy bajo. Las herramientas tradicionales de gestión de datos junto con las técnicas estadísticas no siempre son adecuadas para analizar ingentes volúmenes de datos.

En el ámbito de la minería de datos, el aprendizaje de conceptos basado en inducción de ejemplos es una tarea de alta complejidad. Éste depende de la topología de los datos a analizar y del volumen de éstos. A medida que el volumen de datos crece, el tiempo necesario para su tratamiento se incrementa de manera exponencial, además de crecer la dificultad de aprendizaje del algoritmo. Existen diversas alternativas para aminorar el tiempo de proceso: una de ellas es la reducción de datos con el consiguiente riesgo de pérdida de precisión del modelo; otra es la mejora de la escalabilidad del algoritmo para abarcar el volumen completo de datos, como es el caso de los algoritmos genéticos distribuidos para

clasificación. En cualquier caso, estos algoritmos también presentan algunas limitaciones cuando los conjuntos de datos son muy grandes.
Por otro lado, en los últimos años la aparición de nuevas técnicas de almacenamiento distribuido [1] ha proporcionado nuevas formas de desarrollar algoritmos paralelos y distribuidos que está permitiendo disminuir el tiempo necesario para tratar millones de datos con maquinas de bajo coste. Este paradigma esta basado principalmente en las funciones MapReduce [1] y el sistema de almacenamiento distribuido HDSF [2].
En este sentido, aunque se han conseguido desarrollar versiones adaptadas a este modelo distribuido de buena parte de los algoritmos de aprendizaje a partir de datos, algunos de los modelos de aprendizaje, como por ejemplo los que utilizan algoritmos genéticos para la extracción de reglas, presentan ciertas dificultades para su implementación debido a la necesidad de acceder repetidamente al conjunto completo de datos para calcular la bondad de las reglas o conjuntos de ellas en el alto número de iteraciones que éste tipo de algoritmos realiza durante su ejecución. El modelo MapReduce permite mejoras en éste ámbito pero no llega a hacer que la programación sea equivalente y es necesario por tanto establecer estrategias de implementación que conserven la calidad de los resultados y mantengan el tiempo de ejecución en niveles competitivos respecto a otras alternativas distribuidas.

Este trabajo repasa los algoritmos genéticos distribuidos más relevantes para la extracción de reglas, describe sus límites respecto a grandes conjuntos de datos y trata de analizar las opciones de implementación sobre el paradigma MapReduce.

La organización de este trabajo es la siguiente: en la segunda sección se revisan los algoritmos genéticos distribuidos para clasificación de la literatura, el paradigma MapReduce y las plataformas y librerías más extendidas para la aplicación al aprendizaje a partir de datos. Asimismo se describen brevemente algunas de las principales propuestas existentes para la implementación de algoritmos genéticos sobre MapReduce . En la tercera sección se proponen varias estrategias de aplicación de este modelo a algoritmos genéticos distribuidos para extracción de reglas de clasificación. El modelo resultante permitiría incrementar la escalabilidad con MapReduce y por otro lado mantener la calidad de la solución final mediante el uso de conjuntos de entrenamiento distribuidos obtenidos al aplicar diferentes estrategias de reducción de datos.

2 Antecedentes

En este apartado se describen las principales referencias de algoritmos genéticos distribuidos para la clasificación y dentro de big data se revisa el modelo MapReduce y propuestas para su uso en la clasificación

2.1 Algoritmos genéticos distribuidos en clasificación

A continuación se van a revisar los principales algoritmos de este área, muy brevemente sus principios y finalmente sus limitaciones y el origen de las mismas.

Una de las principales referencias en este ámbito es REGAL [17]. Éste algoritmo propone una división de datos en nodos que contienen algoritmos genéticos para el aprendizaje de reglas que pueden no ser validas para todo el conjunto de datos. Posteriormente refina dichas reglas asignando cada una de ellas y sus datos asociados a diferentes nodos. REGAL-TC [18] mejora la precisión de REGAL a través de un ponderado de contra-ejemplos. Esta técnica permite dirigir el esfuerzo de aprendizaje en las etapas de refinamiento del modelo hacia aquellas reglas que son difíciles mejorando la precisión del modelo generado. Now-GNet [19] propone una mejora sobre el mismo esquema proponiendo un buffer de reglas a evaluar que separa las funciones de evaluación del individuo y de algoritmo genético, esto permite un mayor grado de paralelización. Estos tres algoritmos tienen en común la presencia de un proceso supervisor síncrono que espera los resultados parciales de los nodos para decidir la distribución de datos en las siguientes tareas. Todos están implementados bajo MPI [3].
EDGAR [20] por su parte utiliza un modelo mixto de nodos aprendedores independientes con un subconjunto de los datos y supervisor central. El sistema es asíncrono optimizando el funcionamiento independiente de los procesadores respecto al supervisor, aunque necesita una copia del conjunto global de datos para evaluar las reglas recibidas y generar un clasificador final.
Si bien los algoritmos genéticos distribuidos para clasificación tienen en su esencia uno de los elementos apropiados para afrontar conjuntos de datos grandes en base a su escalabilidad, cuando los conjuntos de datos son del orden de magnitud de los considerados en Big Data, encuentran dos tipos de dificultades:

1. Suelen necesitar tener el conjunto completo de datos a tratar en la memoria principal (en el nodo principal o asignados a alguna regla de amplia cobertura), y por tanto, cuando el conjunto excede a ésta, no son aplicables.
2. En muchos casos también se suele observar a partir de cierto tamaño un deterioro progresivo de la calidad de los resultados proveniente probablemente del excesivo particionamiento del conjunto de datos en las fases iniciales de aprendizaje.

Estas limitaciones nos invitan a considerar modelos híbridos con MapReduce, que se discutirán en la sección 3.

2.2 MapReduce

El modelo MapReduce [1] permite un modelo de diseño paralelo basado en la división de datos. Este modelo se basa en la existencia de conjunto de datos distribuidos que permiten una alta escalabilidad de datos y la codificación de tareas sobre estos basados fundamentalmente en dos operaciones Map y Reduce. Estas dos operaciones ocultan la complejidad del hardware paralelo y permite simplificar enormemente el diseño de algoritmos paralelos.

Aunque inicialmente este modelo fue pensado para procesar grandes cantidades de datos distribuidas (Map) que luego tienen una agregación posterior para producir unos resultados (Reduce), la simplicidad de las operaciones permite utilizar este modelo de diseño incluso cuando no hay grandes volúmenes de datos, permitiendo estructurar algoritmos paralelos que se comunican a través de la agrupación de valores entorno a nuevos conjuntos de claves.

El modelo MPI [3], el cual permite realizar diseños paralelos y ha sido el estándar de facto durante décadas, a medida que crece el número de procesos, la heterogeneidad de la red y los procesos, el modelo se vuelve complejo y poco resistente a fallos. Por su parte, MapReduce no es siempre una alternativa a un desarrollo paralelo dado que hay que reformular el algoritmo en términos de estas dos operaciones y que la relación entre ellas no genere excesivos accesos a disco o malas configuraciones de comunicación que generen cuellos de botella.

Plataformas El principal cuello de botella en el paradigma MapReduce se produce en la parte de transformación inicial de datos, que requiere de un acceso al almacenamiento físico. La primera implementación extendida de este modelo es Hadoop [2]. Ésta sigue siendo usada ampliamente por la comunidad especialmente en su versión 2 que simplifica el diseño de algoritmos paralelos. Sin embargo la necesidad de acceder repetidamente a ciertos datos no esta resuelta en Hadoop y han surgido otras opciones como Spark [4] y Haloop [5] que permiten realizar un caché de datos en memoria para evitar que se acceda a la capa física. Aunque la mejora de rendimiento es considerable, del orden 10 a 30 veces más rápido, la transformación de algoritmos a las primitivas MapReduce sigue siendo un punto crítico de diseño.

Librerías de Aprendizaje Automático. En los últimos años han surgido librerías que permiten la codificación de algoritmos de aprendizaje de alto nivel con un conocimiento mínimo de MapReduce. MLBASE [6] y MAHOUT [21], permiten realizar algoritmos de clasificación, *clustering* y otros modelos de aprendizaje principalmente sobre Hadoop. MAHOUT ha dejado de utilizar Hadoop a partir de

2014 para implementar más eficientemente el problema del acceso repetido a datos. MLLIB [7] es una librería equivalente sobre la plataforma SPARK.

2.3 MapReduce en el ámbito de la clasificación

Algunos trabajos en el ámbito de la clasificación utilizan MapReduce para disminuir la complejidad del modelo realizando un preprocesado de los datos iniciales. Bien para compensar las clases no balanceadas [7], bien para discretizar el conjunto de datos [22] o mediante selección de instancias [9] o características [10] para poder trabajar directamente con el conjunto de datos reducido.

Dentro del aprendizaje de modelos de clasificación, hay técnicas que se prestan mejor que otras a este paradigma porque no necesitan de un acceso repetido a los datos. La generación de clasificadores con Random Forest [7] implementados bajo MapReduce realiza la evaluación de múltiples árboles de decisión eficientemente. En [11] se aplican varias técnicas implementadas sobre MapReduce, en primer lugar realiza una selección de características evolutivo, posteriormente aplica el modelo de [7] de balanceado y clasificador basado en Random Forest para generar el modelo global del clasificador.

Otra alternativa es la generación de reglas utilizando los ejemplos como semillas que posteriormente formarán el clasificador. En [12] se aplica Map para generar reglas en base a un conjunto de etiquetas lingüísticas que son evaluadas en la fase de construcción y posteriormente seleccionadas en la fase Reduce para generar el conjunto de reglas final teniendo en cuenta el coste de la clasificación en caso de existir clases no balanceadas.

2.4 Implementación de Algoritmos Genéticos en MapReduce

En [13] se propone una estrategia de transformación genérica de algoritmos genéticos con las primitivas MapReduce, que encapsulan cada iteración de un algoritmo genético en un trabajo MapReduce, orientado a realizar un algoritmo genético equivalente con calculo distribuido del fitness. Este esquema permite una distribución de la carga eficiente entre los distintos reduces que agregan los resultados. Una de las formas más utilizadas en la paralelización de algoritmos genéticos es la estructura en islas [14]. Hay varias alternativas en MapReduce que facilitan la implementación de este modelo. Elephant56 [16] ofrece un *framework* donde las primitivas utilizan directamente el modelo de nodos islas como base de paralelización del algoritmo. MRPGA [15] propone primitivas especificas para

distribuir evaluaciones, para seleccionar elementos y para conectar diferentes procesos que permiten mayor nivel de detalle en la comunicación entre procesos

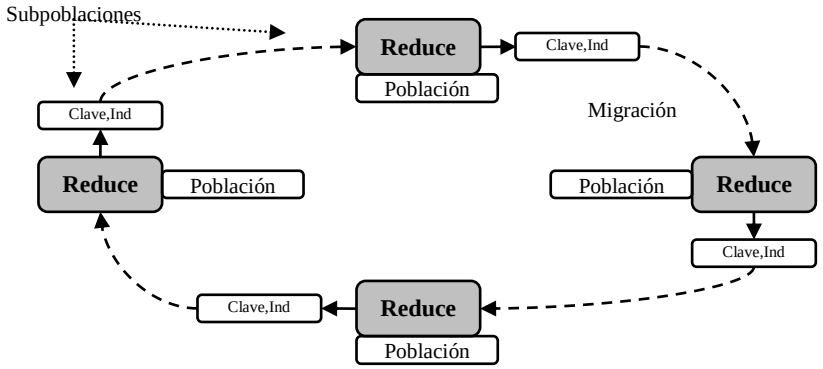

Fig 2.1 Utilización de Reduce para la implementación de islas

3 Propuestas de implementación sobre Algoritmos Genéticos Distribuidos de extracción de reglas de clasificación

Los algoritmos genéticos distribuidos para extracción de reglas permiten escalar el conjunto de datos de entrenamiento manteniendo el equilibrio entre exploración y calidad de la solución. Sin embargo cuando el conjunto de datos sobrepasa ciertos límites el modelo puede presentar problemas de rendimiento por los límites de memoria de los procesadores o bien de calidad de la solución por el excesivo particionamiento de los datos en procesos. En este apartado vamos a analizar brevemente el impacto del modelo MapReduce sobre la evaluación del *fitness* en un algoritmo genético Secuencial. Posteriormente propondremos alternativas para la mejora de las algunas de la propuestas existentes de algoritmos genéticos distribuidos para la clasificación que permitirían mantener las características del modelo distribuido con cualquier volumen de datos.

3.1 Modelo secuencial paralelizado sobre MapReduce

Como se ha comentado anteriormente, los algoritmos genéticos para la extracción de reglas son muy sensibles al tamaño del conjunto de entrenamiento, tanto en rendimiento como en capacidad de aprendizaje. La principal estrategia de paralelización de un algoritmo genético manteniendo la estructura secuencial del mismo es la de la paralelización del cálculo del *fitness* del individuo [13]. Al

aplicar esta estrategia a un algoritmo de aprendizaje de reglas en MapReduce el algoritmo se beneficiaría de no tener un límite de datos determinado, dado que la operación de evaluación de una regla se puede realizar en un solo paso MapReduce.
Sin embargo cada operación Map tiene asociado un tiempo de latencia relativamente alto en el entorno de los milisegundos debido a la comunicación por red más el acceso al medio físico si accede a la base de datos (en Spark puede almacenarse en memoria distribuida). En este caso el tiempo de ejecución del algoritmo es relativo al numero de iteraciones (*k*) y a la latencia de acceso al medio físico mas la comunicación de red (*l*). Aún cuando el rendimiento pueda ser inferior al del modelo secuencial en un procesador, hay otros motivos por los que puede ser conveniente utilizar MapReduce:

- A medida que la magnitud de datos de entrenamiento crece es más difícil almacenarlo de manera integra en un sólo sistema.
- El coste económico de instanciar un conjunto de servidores virtuales que soporten una ejecución sobre el paradigma Hadoop es muy inferior al de una maquina (o servidor virtual) con la memoria equivalente para contener el conjunto total de los datos.
- La evaluación de cada regla contra un conjunto elevado de elementos en un solo sistema requiere un algoritmo de orden n (cada elemento por las n reglas del conjunto de individuos que conforman la población del genético). Aún siendo orden n, el tiempo crecerá de forma pareja al aumento de los datos mientras que en MapReduce el tiempo se mantiene constante. En la formula (1), se expresa la relación entre el coste de mantener el conjunto de datos de manera monolítica en un servidor o tener el *fitness* distribuido sobre MapReduce) siendo *c* coste en memoria principal en milisegundos, *n* número de datos y *l* la latencia:

(1) $c*n > l$ por tanto $n > l/c$

A partir de este valor *n* de datos, el sistema sobre MapReduce ofrecerá mejoras en el rendimiento.

Como hemos comentado en el apartado 2, otra estrategia para disminuir el coste del aprendizaje asociado a la cardinalidad es la reducción de datos (discretización, selección de características, selección de instancias…) para que el conjunto de datos resultante pueda ser tratado directamente en memoria principal.

La selección de instancias y la discretización sin repetición de elementos, tienen el efecto de representar en un solo ejemplo a varios elementos del conjunto original. Generalmente en el preprocesado de datos no se añade información en forma de metadatos para ayudar al proceso de aprendizaje a discernir que datos son más representativos. Nosotros proponemos añadir metadatos a los datos de

entrenamiento reducidos para ajustar el valor de clasificación de las reglas. El algoritmo puede entonces calcular la función de bondad (*fitness* en adelante) de un elemento teniendo en cuenta el número real de instancias representadas.

(2) Fitness =casosTotales* e^(1/casos negativos*)

En (2) se muestra una posible representación de *fitness* de una regla, donde el número casos totales y casos negativos, son un sumatorio ponderado de los casos reales a los que cubre y los casos negativos en los que la regla no predice la clase correcta. Esta formula permite mantener el coste de la clasificación errónea respecto a los datos iniciales representados manteniendo un conjunto reducido de datos. Cuando se trabaja con clases no balanceadas, se puede incluir un peso de fallo en la clasificación de las clases que represente la proporción real entre las reglas. Durante el proceso de disminución del modelo este peso puede ser calculado para formar parte de la ponderación de los casos representados respecto a los originales.

3.2 Algoritmos Genéticos Distribuidos de Clasificación sobre MapReduce

Los algoritmos distribuidos tienen como principal característica que la evaluación de los individuos y los datos asociados a ellos se realiza en procesadores paralelos. Este apartado recoge las posibles estrategias de implementación sobre las principales referencias de algoritmos distribuidos de aprendizaje de reglas en base a las operaciones básicas Map y Reduce.

Como se ha comentado anteriormente, algunas de las principales referencias de algoritmos genéticos distribuidos para la clasificación [17][18][19] son implementables en MapReduce de manera equivalente. El modelo MPI de sincronismo puede ser implementado con Nodos Reduce y las evaluaciones de los individuos con Map. Sin embargo ello conlleva una perdida de rendimiento asociada al acceso físico para la evaluación del *fitness* de una regla, que según (1), tendrá una cota superior de datos a partir de la cual va a tener un comportamiento constante con el aumento de los datos a tratar.

EDGAR [20] separa el proceso de aprendizaje en los nodos de la tarea de validación de las regla en un supervisor con el total de los datos de entrenamiento. Esta implementación podría realizarse en MapReduce sin perdida de rendimiento respecto al modelo original en memoria principal, realizando la operación de validación global mientras que los nodos están realizando el aprendizaje de nuevas

reglas, ya que éstos no necesitan esperar a otros nodos o al procesador maestro para generar reglas o compartir datos y reglas con nodos.

Por otro lado la reducción del conjunto de datos inicial podría permitir a estos algoritmos utilizar los nodos de aprendizaje directamente en memoria principal y utilizar los datos totales en la generación del clasificador final. Si bien esto es posible para todas las referencias, en [16][17][18] la asignación de una regla y los datos globales asociados a la misma para su refinamiento no aportaría una gran mejora del rendimiento. Sin embargo [20] separa estas dos funciones por lo que planteamos dos alternativas en base a la función realizada por el nodo supervisor y los nodos de aprendizaje con los datos preprocesados.

Por un lado proponemos utilizar en los nodos aprendedores un conjunto de datos reducido (discretización, selección de instancias o características) que envían reglas al supervisor que valida y genera la solución global con MapReduce con el conjunto completo de entrenamiento. Esta validación minimiza la perdida de calidad producida por el proceso de reducción de datos en los aprendedores.

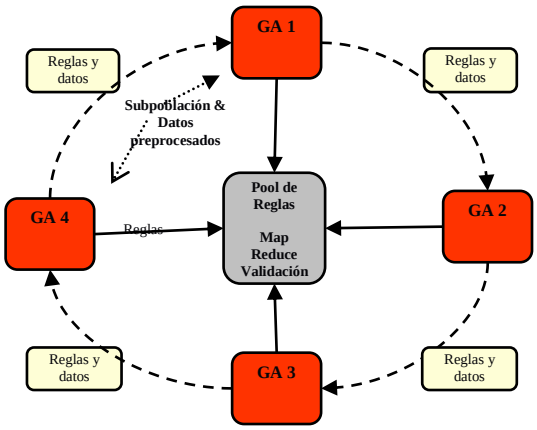

Fig 3.1 Modelo Mixto MapReduce – procesamiento local sobre EDGAR

Por otro lado esta alternativa se puede mejorar utilizando reducción de datos con distintas técnicas (distintos discretizadores, selección de instancias etc.). El clasificador final estará compuesto por la integración de las mejores reglas validadas de forma global minimizando de esta forma la perdida de calidad que se pueda haber producido en el proceso de una sola técnica de reducción de datos.

4 Conclusiones

El modelo MapReduce permite abarcar conjuntos de datos que no son tratables directamente en memoria principal, sin embargo no tiene un rendimiento comparable a las implementaciones que trabajan con todos los datos en memoria principal. La hibridación de técnicas de disminución del conjunto de datos permite el tratamiento en memoria, pero a cambio puede afectar a la calidad del modelo generado. Este trabajo propone alternativas basadas en la colaboración de modelos locales de menor magnitud con la validación global usando MapReduce que permitiría mantener los tiempos de ejecución abarcando un conjunto de datos mucho mayor sin que la calidad del modelo se vea afectada.

Referencias

[1] Dean J., Ghemawat S.: MapReduce: Simplified Data Processing on Large Clusters. Commun ACM (2008) 51:107–113
[2] White T.: Hadoop: The Definitive Guide. 1st ed.Sebastopol, CA: O'Reilly (2009)
[3] Snir M., Otto S.: MPI-The Complete Reference: The MPI Core. Cambridge, MA: MIT Press (1998)
[4] Zaharia M., Chowdhury M., Franklin M.J., Shenker S., Stoica I.: Spark: Cluster Computing with Working Sets. HotCloud, Boston, MA (2010) 1–7
[5] Bu Y., Howe B., Balazinska M., Ernst M.D.: The Haloop Approach to Large-Scale Iterative Data Analysis. PVLDB (2012) 21:169–190
[6] Kraska T., Talwalkar A., Duchi J., Griffith R., Franklin M., Jordan M.: Mlbase: a Distributed Machine Learning System. Conference on Innovative Data Systems Research, Asilomar, CA (2013) 1–7
[7] Río S., López S., Benítez J.M., Herrera F.: On The Use of MapReduce for Imbalanced Big Data using Random Forest. Information Sciences 285 (2014) 112-137
[8] Zhang Y., Yu J., Wang J.: Parallel Implementation of Chi2 Algorithm in MapReduce Framework. Human Centered Computing, Lecture Notes in Computer Science (2015) Volume 8944, 890-899
[9] Triguero I., Peralta D., Bacardit J., García S., Herrera F.: MRPR: A MapReduce Solution for Prototype Reduction in Big Data Classification. Neurocomputing 150 (2015) 331-345
[10] Peralta S., del Río S., Ramírez-Gallego S., Triguero I., Benítez J.M., Herrera F.: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach. Mathematical Problems In Engineering (2015). Article ID 246139, in press
[11] Triguero I., del Río S., López V., Bacardit J., Benítez J.M., Herrera F.: ROSEFW-RF: The Winner Algorithm for the ECBDL'14 Big Data Competition: An Extremely Imbalanced Big Data Bioinformatics Problem . Knowledge-Based Systems Vol. 87 (2015) 69-79
[12] del Río S., López V., Benítez J.M., Herrera F.: A MapReduce Approach to Address Big Data Classification Problems Based on the Fusion of Linguistic Fuzzy Rules. International Journal of Computational Intelligence Systems (2015) 422-437
[13] A. Verma, X. Llor`a, D. Goldberg, R. Campbell: Scaling Genetic Algorithms Using MapReduce. Proceedings of International Conference on Intelligent Systems Design and Applications (2009) 13–18
[14] Cantu-Paz E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic publishers (2000)
[15] Jin C., Vecchiola C., Buyya R.: MRPGA: An Extension Of MapReduce for Parallelizing Genetic Algorithms. IEEE Fourth International Conference on eScience (2008). 214–221
[16] Geronimo D., Ferrucci L.F., Murolo A., Sarro F.: A Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of Junit Test Suites. Software Testing, Verification and Validation (ICST) IEEE Fifth International Conference (2012) 785-793
[17] Giordana A., Neri F.: Search-Intensive Concept Induction. Evolutionary Computation (1995) 375–416
[18] Lopez L.I., Bardallo J.M., De Vega M.A., Peregrin A.: Regaltc: A Distributed Genetic Algorithm for Concept learning based on regal and the treatment of counterexamples. Soft Computing Volumen 15(7) (2011) 1389–1403
[19] Anglano C., Botta M.: NOW G-Net: Learning Classification Programs on Networks of Workstations. IEEE Transactions on Evolutionary Computation (2002) 463-480
[20] Rodríguez M., Escalante D.M., Peregrín A.: Edgar: an Efficient Distributed Genetic Algorithm for Rule Extraction. Applied Soft Computing Vol. 11(1) (2011) 733–743
[21] Owen S., Anil R., Dunning T., Friedman E.: Mahout in Action 1st ed. Manning (2011)

[22] Ramirez S.: Distributed Minimum Description lenght Discretizer for Spark. https://github.com/sramirez/ spark-MDLP-discretization (2015)