Clasificación multi-etiqueta utilizando computación distribuida

Juan Manuel Rodriguez*[1], Alejandro Zunino*[2], Daniela Godoy*[3] and Cristian Mateos*[4]

*ISISTAN, UNICEN-CONICET
Campus Universitario - Paraje Arroyo Seco, Tandil, Buenos Aires , Argentina

[1]juanmanuel.rodriguez@isistan.unicen.edu.ar
[2]alejandro.zunino@isistan.unicen.edu.ar
[3]daniela.godoy@isistan.unicen.edu.ar
[4]cmateos2006@gmail.com

***Resumen*—Las técnicas de clasificación multi-etiqueta fueron desarrolladas para problemas donde los objetos pueden estar asociados a distintas etiquetas disjuntas, por ejemplo las áreas de un artículo científico. Sin embargo, estas técnicas puede ser computacionalmente costosas, lo que dificulta su aplicabilidad en dominios reales. En este trabajo se presenta un enfoque para acelerar un algoritmo de clasificación multi-etiqueta llamado *Binary Relevance*. En este algoritmo, la complejidad de crear un modelo de clasificación crece linealmente con el número de etiquetas que pueden ser asignadas a una instancia a clasificar, por lo que se propuso un enfoque para utilizar pequeños clústers de computadoras para gestionar el cómputo de la fase de entrenamiento del clasificador. El enfoque fue probado con 7 conjuntos de datos de entrenamiento con 81 etiquetas asociadas y más de un cuarto de millón de instancias para entrenar el clasificador. Los resultados mostraron que se aceleró de manera lineal el tiempo de entrenamiento a medida se agregaron nodos computacionales al clúster.**

***Abstract*— Multi-label classification techniques have been developed for problems where objects can be associated to several disjoint labels. However, these techniques tend to be computational complex, which difficult their applicability in practice. Therefore, they might be unsuitable for large problems. This work presents an approach to accelerate one of such techniques, called Binary Relevance, using small computational clusters. This approach was tested using 7 data-sets with 81 associated labels and more than a quarter million training instances. Experiments have shown a linear speed-up on the number of computing nodes added to the cluster.**

I. Introducción

Las técnicas tradicionales de aprendizaje automático de clasificación están diseñadas para problemas donde cada elemento a clasificar, llamados *instancias*, pueden tener solo una clase. Sin embargo, en la realidad existen problemas de clasificación donde una misma instancia puede pertenecer a distintas clases disjuntas, llamadas etiquetas [1]. Un claro ejemplo de ésto son los artículos científicos debido a que pueden contribuir a distintas áreas del conocimiento, o el caso de una composición musical que corresponde a varios géneros.

Formalmente, una instancia i está representada por $v_i = (x_1, x_2, ..., x_m) \in X$ donde X es el espacio de instancias posibles, mientras que las etiquetas que se asocian a una instancia son un subconjunto $Y_i \subseteq Y$ de todas las etiquetas posibles, con $Y = \{y_1, y_2, ..., y_n\}$. En este contexto, el problema de clasificación multi-etiquetas es encontrar una función $f : X \longrightarrow 2^Y$ a partir de un conjunto de instancias de entrenamiento $E = \{(v_j, Y_j)|1 \leq j \leq d\}$, donde d es el tamaño de X. Para una nueva instancia t, la función $f(v_t)$ predice un subconjunto de etiquetas $Y_t \subseteq Y$ [2].

Una de las técnicas más conocidas para clasificación multi-etiquetas es *Binary Relevance* (BR), que consiste en transformar el problema en múltiples problemas de clasificación tradicional, uno por cada etiqueta de Y. Es decir, BR transforma el problema en varios problemas de clasificación binaria donde cada clasificador es el encargado de decidir si a una instancia le corresponde o no una etiqueta. Básicamente, para predecir el conjunto de etiquetas de una instancia nueva, BR combina los resultados de diferentes clasificadores asociados a las etiquetas. Finalmente cabe aclarar que BR no condiciona el tipo de clasificador que se utiliza para aprender cada etiqueta por separado, por lo que éste puede ser combinado con cualquiera de las técnicas tradicionales como *Sequential Minimal Optimization* (SMO) [3] o *Naive Bayes* [4].

A pesar de su simplicidad, la utilización de BR puede verse perjudicada por su gran costo computacional. En un trabajo reciente [5] se evidencia este hecho debido a que ciertos experimentos utilizando BR no pudieron ser completados a pesar de haber corrido por más de una semana en un servidor Linux con dos procesadores Intel Quad-Corecorriendo a 2.5 GHz y 64 GB de RAM. Uno de las potenciales causas es que las bibliotecas de clasificación multi-etiquetas usadas no aprovechan totalmente los recursos de los sistemas multi-procesador. En particular, la popular biblioteca Mulan [6], utilizada por [5], es completamente secuencial. Esto significa que la mayoría de los 8 núcleos disponibles para el experimento no se han aprovechado.

En este trabajo se propone el re-diseño e implementación del método de clasificación multi-etiqueta BR de Mulan no solo para utilizar varios núcleos disponibles en un servidor, sino para aprovechar los núcleos disponibles en un clúster de computadoras. De esta manera, se pretende escalar las capacidades de BR de forma significativa agregando poder de cómputo.

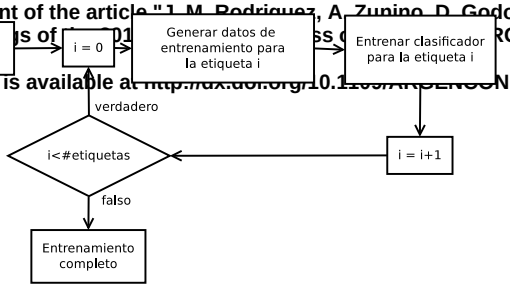

Figura 1. Implementación de *Binary Relevance* en Mulan

El resto del trabajo se estructura de la siguiente manera. En la Sección II se discuten trabajos relacionados. La Sección III presenta el enfoque utilizado para gestionar BR de manera distribuida en un clúster de computadoras. La Sección IV discute los resultados experimentales. Finalmente, la Sección V concluye el trabajo y discute líneas de trabajo futuro en el área de clasificación multi-etiqueta utilizando ambientes de computación distribuida.

II. Trabajos relacionados

Recientemente, se han propuesto trabajos que explotan técnicas de computación distribuida en clasificación multi-etiqueta. Si bien el foco en el área sigue siendo incrementar la eficiencia de las técnicas en términos de métricas tales como precisión, *recall* y *Hamming loss*, hay un fuerte consenso [5], [7], [8] en que el tiempo de entrenamiento y clasificación es un aspecto a considerar, sobre todo en presencia de conjuntos de datos grandes.

Algorítmicamente, existen dos métodos para resolver problemas de clasificación multi-etiqueta: adaptación y transformación. La adaptación apunta a extender clasificadores tradicionales para clasificación multi-etiqueta, siendo Multi-label C4.5 [9] y ML-KNN [10] ejemplos de estos clasificadores. Por otro lado, la transformación apunta a transformar el problema multi-etiqueta a varios problemas menores de clasificación tradicional cuyos resultados se agregan para dar una solución global. Precisamente, BR [8] se basa en transformar el conjunto de datos de entrenamiento en conjuntos de datos de entrenamientos para una sola etiqueta. Es decir, crea tantos conjuntos de datos de entrenamiento como etiquetas tengan los datos originales. Estos nuevos conjuntos son binarios, por lo que se puede utilizar un clasificador binario tradicional, como SMO, para predecir cada etiqueta. La Figura 1 describe esquemáticamente como se entrena un clasificador BR. En BR clasificar una nueva instancia es simplemente retornar las etiquetas que dieron positivo de acuerdo a los clasificadores binarios. En particular, la Figura 1 describe la implementación secuencial de BR que brinda la biblioteca Mulan [6]. Mulan es una extensión de la ampliamente utilizada biblioteca Weka [11], la cual implementa una amplia variedad de algoritmos de aprendizaje automático.

En general, el método de transformación y en particular el algoritmo BR es la forma de clasificación multi-etiqueta más utilizada [12], [13]. Desde una perspectiva distribuida, BR puede ser representado como una aplicación maestro-esclavo [14], donde varios esclavos (clasificadores tradicionales) pueden ser ejecutados independientemente, y el maestro (regla 4FK) controla los resultados de los esclavos. En este sentido, existen trabajos preliminares que gestionan los esclavos en paralelo en un ambiente distribuido para mejorar el rendimiento de la clasificación. Por ejemplo, [12] implementa un framework basado en *MapReduce* y BR para problemas de clasificación de texto. Un punto débil del trabajo es que los requerimientos computacionales propios del framework *MapReduce* (memoria y tiempo de ejecución) no han sido determinados empíricamente aún. El algoritmo HOMER [13] divide el conjunto de etiquetas de entrada en conjuntos disjuntos por similitud, representados como una estructura arborescente. Luego, la clasificación se realiza en paralelo por varios esclavos, cada uno utilizando un subárbol de la estructura. Finalmente, *scikit-learn* [15] es una biblioteca que incluye algoritmos de clasificación multi-etiqueta con un incipiente soporte de paralelismo y distribución. En este sentido, agregar paralelismo a BR permitiría realizar pruebas comparativas contra estos algoritmos y determinar en qué situaciones conviene utilizar uno u otro.

III. BR paralelo: Enfoque

En este trabajo se presenta una adaptación del algoritmo BR originalmente presentado en Mulan [6] para poder utilizarlo eficientemente en clústers computacionales. La implementación de Mulan del algoritmo BR es completamente secuencial, es decir, se entrena un clasificador de etiqueta a la vez. Aunque esto reduce el consumo de memoria porque solo se debe tener una copia del conjunto de datos de entrenamiento en memoria, el mecanismo no aprovecha las capacidades de los sistemas multi-núcleo actuales. Nuestra propuesta no solo toma ventaja de estas capacidades, sino que además permite escalarlas mediante el agregado horizontal de nodos de cómputo.

El enfoque propuesto sigue una arquitectura *maestro-esclavo* en donde un nodo, el maestro, es el encargado de orquestar a los otros nodos para que entrenen cada uno de los clasificadores binarios. En la Figura 2 se presentan esquemáticamente los pasos llevados a cabo para entrenar el clasificador BR paralelo. Primeramente, al inicio del entrenamiento el nodo master transfiere el conjunto de datos de entrenamiento a cada uno de los nodos del clúster. Estos datos contienen para todas las instancias de entrenamiento sus atributos y etiquetas. Una vez transferidos los datos, se comienza con el entrenamiento de cada uno de los clasificadores. Para esto, cada nodo esclavo del clúster que posee capacidad de cómputo disponible le informa al nodo maestro que puede entrenar un clasificador. Como el proceso de entrenamiento es intensivo en uso de CPU, cada nodo entrena tantos clasificadores como núcleos tenga disponibles.

Para soportar la comunicación entre los distintos nodos del clúster, la implementación aquí descripta utiliza JGroups[1], una biblioteca para implementación de sistemas que requieren multicast confiable, es decir redes que permiten enviar la misma información a diversos nodos. Adicionalmente, JGroups se organiza internamente con una arquitectura de capas de protocolos que permiten una gran flexibilidad. Básicamente, se puede definir desde que tipo de

[1] JGroups: http://www.jgroups.org/

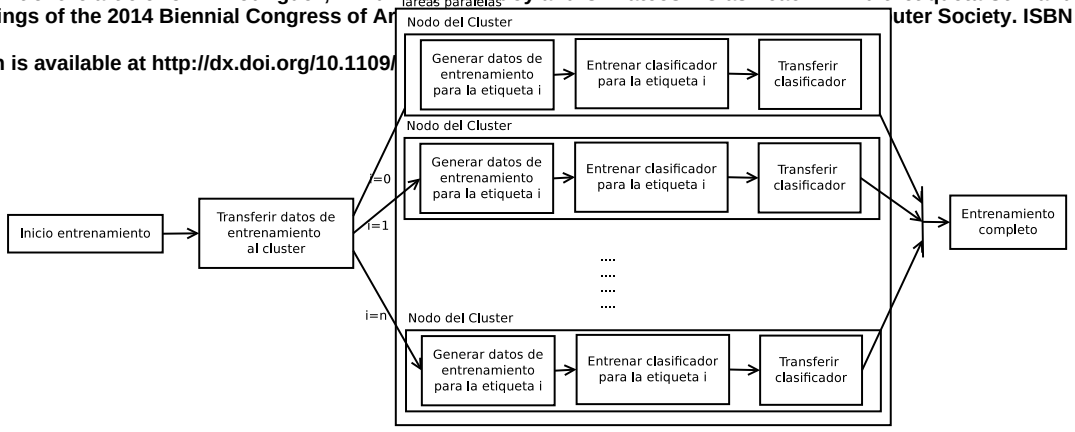

Figura 2. BR para clústers

comunicación se utilizará, por ejemplo TCP o UDP, hasta servicios de alto nivel como semáforos distribuidos.

En particular, la implementación del BR para clúster utiliza la capacidad *multicast* para copiar los datos de entrenamiento a los nodos del clúster. Esto significa que si la tecnología de red subyacente lo permite, por ejemplo IP *multicast* sobre Ethernet, el tiempo necesario para copiar los datos de entrenamiento a los nodos del clúster es independiente de la cantidad de nodos que existan en el clúster.

Otra ventaja de JGroups es que permite fácilmente transmitir objetos Java entre distintos nodos. La única restricción es que las clases de los objetos implementen la interfaz `java.io.Serializable`[2]. Esta interfaz no define ningún método, pero es utilizada como manera de identificar que los objetos pertenecientes a esa clase puede ser aplanados a un conjunto de bytes, lo que es útil por ejemplo para enviar por red o almacenar en disco una instancia de la clase. Este mecanismo de convertir objetos en una lista de bytes se conoce como serialización. En este sentido, Mulan hace amplia utilización de esta interfaz, no para almacenar o enviar objetos por red, sino para hacer copias en profundidad de objetos. Sin embargo, esto permitió implementar BR para clústers sin tener que realizar modificaciones a las clases que definen los clasificadores ni a las que definen cómo se almacena las instancias de entrenamiento.

IV. Resultados experimentales

Con el objetivo de evaluar el enfoque se utilizó un clúster de hasta 6 equipos con procesadores AMD FX-6100 de 6 núcleos cada uno con 16 GB de RAM como nodos esclavo, totalizando 36 núcleos. El nodo maestro fue un servidor Intel i7-3820 con 32 GB de RAM. Todos los nodos corrían Linux Ubuntu 13.10 y Java 7. El clúster está interconectado con una red Ethernet de un gigabit. Con respecto a JGroups se utilizó la configuración por defecto de comunicación basada en UDP IP multicast.

En los experimentos se utilizaron 7 conjuntos de datos de entrenamiento que fueron generados a partir de imágenes reales [16]. Estos datos poseen un total de 81 etiquetas y 269.648 instancias. La diferencia entre los diferentes conjuntos de datos son los atributos utilizados para representar cada instancia. El Cuadro I presenta una descripción de los datos de prueba. Las primeras tres columnas presentan información propia de los conjuntos de datos utilizados para las pruebas: sus nombres, el número de atributos, y el porcentaje de valores que son cero (nulos). Las últimas dos describen información de la representación utilizada internamente por Mulan. En la cuarta columna se describe si el conjunto de datos se encuentra cargado en memoria utilizando representación rala o densa, es decir si las instancias están representadas por objetos del tipo `SparseInstance`[3] o `DenseInstance`[4], respectivamente. Finalmente, la última columna presenta el tamaño de la representación cuando ésta es serializada mediante los mecanismos propios de Java.

Cuadro II
Tiempo total de corrida Mulan

Conjuntos de datos	Tiempo total (hh:mm:ss)
All tags	09:06:30
LLF Normalized BoW	02:23:53
LLF Normalized CH	00:17:04
LLF Normalized CM55	01:08:08
LLF Normalized CORR	00:37:42
LLF Normalized EDH	00:19:33
LLF Normalized WT	00:35:04

Metodológicamente, la experimentación consistió comparar el tiempo requerido para cargar los conjuntos de datos desde los archivos, crear el clasificador multi-etiqueta y entrenarlo. El clasificador BR utilizó como clasificador base Naive Bayes [4]. Luego, se compararon los tiempos requeridos por la implementación original de Mulan con los tiempos requeridos por la implementación presentada en este trabajo, utilizando entre 1 y 6 nodos esclavo. Es importante destacar que el entrenamiento fue ejecutado en los nodos esclavo. El Cuadro II presenta el tiempo requerido para entrenar un clasificador utilizando la implementación original secuencial del Mulan en un equipo con las características de los nodos esclavo. Es importante considerar que cuando se evalúa un clasificador se deben entrenar

[2]Interfaz Serializable: http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

[3]SparseInstance: http://weka.sourceforge.net/doc.dev/weka/core/SparseInstance.html

[4]DenseInstance: http://weka.sourceforge.net/doc.dev/weka/core/DenseInstance.html

Cuadro I
DESCRIPCIÓN DE LOS DATOS DE PRUEBA

Nombre	# Atributos	% valores nulos	Representación	Tamaño serializado
All tags	1.002	99,14 %	Rala	40 Mb
LLF Normalized BoW	500	60,30 %	Densa	1.203 Mb
LLF Normalized CH	64	54,57 %	Densa	306 Mb
LLF Normalized CM55	225	25,86 %	Densa	637 Mb
LLF Normalized CORR	144	35,17 %	Densa	470 Mb
LLF Normalized EDH	73	51,38 %	Densa	324 Mb
LLF Normalized WT	128	37,86 %	Densa	437 Mb

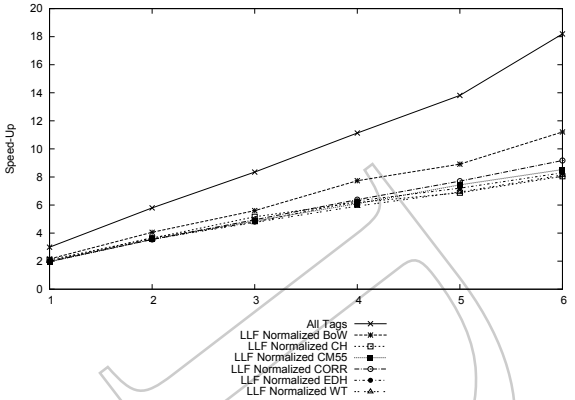

Figura 3. Speed-Up obtenido

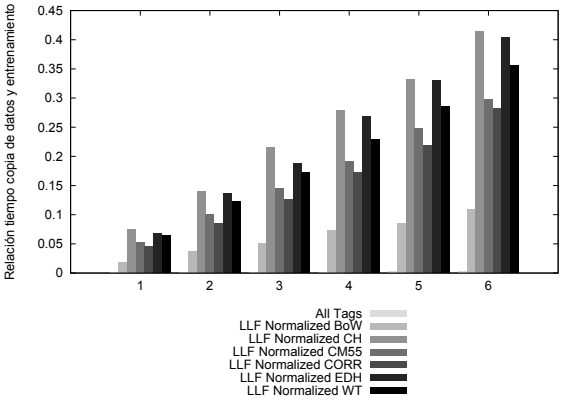

Figura 4. Relación tiempo de copia de datos sobre tiempo de entrenamiento

varias instancias del mismo, en este sentido los tiempos presentados en el Cuadro II se verían multiplicados. Si adicionalmente se utilizara otro clasificador que no sea tan rápido como Naive Bayes, los tiempos de evaluación se elevarían considerablemente. Esto se evidencia en [5], donde se utilizo BR con SMO y ciertos experimentos no pudieron ser terminados a pesar de haber estado corriendo por más de una semana en un servidor de alta prestaciones. Superar esta limitación es el principal objetivo del presente trabajo.

La Figura 3 presenta la aceleración obtenida con nuestro esquema utilizando clústers desde 1 hasta 6 nodos esclavo. Se puede observar que la aceleración crece linealmente con respecto de la cantidad de nodos utilizados. Se puede esperar que este crecimiento continúe hasta que el número de núcleos disponibles sea igual al número de etiquetas en los conjuntos de datos. En ese momento agregar más núcleos no tendría efecto debido a que no se podrían asignar más tareas a los mismos. Con respecto a la aceleración obtenida, uno de los principales obstáculos es el tiempo requerido para serializar y enviar los datos a los nodos del clúster. Esto hace que la pendiente de la curva de aceleración sea menor a uno. De acuerdo a los resultados experimentales, la aceleración más grande se obtiene cuando se utiliza la representación rala. Esto se puede deber a que acceder al valor de un atributo en una representación rala requiere procesamiento para determinar si es cero o no, mientras que en la representación densa acceder a un valor es equivalente a acceder a una posición de memoria.

En la Figura 4 se presenta la relación del tiempo requerido para enviar los datos al clúster para los respectivos cómputos y el tiempo total desde que se comienzan a enviar los datos hasta que se tiene el clasificador BR completamente entrenado. A medida que el valor de esta relación crece,

Cuadro III
TIEMPO DE ENVÍO DE DATOS

Conjuntos de datos	Tiempo promedio (mm:ss)	%Desvío estándar
All tags	00:04,14	1,39 %
LLF Normalized BoW	01:13,23	3,89 %
LLF Normalized CH	00:32,66	2,47 %
LLF Normalized CM55	01:41,42	0,73 %
LLF Normalized CORR	00:47,92	1,33 %
LLF Normalized EDH	00:36,01	1,72 %
LLF Normalized WT	01:01,54	0,96 %

significa que se pasa más tiempo serializando y enviando datos que procesando para entrenar los clasificadores binarios. Es importante destacar que los tiempos de envío de datos, presentados en el Cuadro III, permanecen casi constantes independientemente del número de nodos esclavo. Es necesario destacar que los tiempos de envío no dependen solamente del tamaño de los datos, sino que también del tiempo que le toma a Java recorrer los objetos para construir su versión serializada. Sin embargo, el tiempo de entrenamiento disminuye debido a que al existir más núcleos se pueden entrenar más clasificadores binarios paralelamente, es decir se puede entrenar más clasificadores binarios por unidad de tiempo.

Finalmente, se validaron los clasificadores entrenados utilizando para ello el sistema de evaluación *crossfold* implementado en Mulan. Al aplicar la evaluación utilizando la implementación de Mulan del clasificador BR y la implementación presentada en este trabajo se obtuvieron los mismos resultados. Esto quiere decir que nuestra implementación genera el mismo modelo de clasificación, pero acelerando el proceso de generación. Para evitar disparidad por la utilización de generadores de números aleatorios, todos ellos fueron inicializados con la misma semilla durante

los experimentos.

V. Conclusiones

De acuerdo a los resultados experimentales, es posible utilizar clúster computacionales para reducir los tiempos de entrenamientos necesarios por BR, el cual es ampliamente utilizado en clasificación multi-etiqueta. En este trabajo se presentaron diversos experimentos que validan el enfoque mediante conjuntos de datos grandes con multiplicidad de etiquetas. En el presente trabajo se han analizado la aceleración obtenida con este enfoque y su el principal cuello de botella, es decir, la transmisión de los datos a los distintos nodos de un clúster.

Este trabajo muestra que es viable la utilización de clústers computacionales para aumentar la velocidad de entrenamiento de clasificadores multi-etiquetas. En particular, el clasificador analizado responde al método de transformación, donde el conjunto de datos de entrenamiento multi-etiqueta es procesado con varios clasificadores tradicionales que se focalizan en una sola etiqueta. Si bien Naive Bayes es una técnica tradicional muy sencilla y difundida, existen otras técnicas que pueden ser también utilizadas para crear múltiples problemas independientes de clasificación tradicional, por lo que sería interesante estudiar el enfoque propuesto en este trabajo en el contexto de otras técnicas tales como SMO.

Otro trabajo futuro es analizar maneras más eficientes de serializar objetos Java debido a que la forma estándar de serializar implementada en Java ha sido reconocida como un potencial punto de mejora [17]. Se espera no solo disminuir el tiempo requerido para transformar la información a enviar a bytes, sino también disminuir el tamaño de los datos a transmitir. Adicionalmente, se planea analizar la implementación actual para detectar el uso de los mecanismos de serialización para la creación de copias de objetos y analizar en qué casos se pueden evitar u optimizar los mecanismos de copia.

Finalmente, se planea analizar los métodos de clasificación multi-etiquetas basados en adaptación de algoritmos tradicionales para optimizarlos mediante la utilización de sistemas distribuidos. Este enfoque probablemente presente otras dificultades, debido que a diferencia de los métodos basados en transformación, no se definen claramente tareas independientes a la hora de entrenar al clasificador.

Referencias

[1] C. Sanden and J. Z. Zhang, "Enhancing multi-label music genre classification through ensemble techniques," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, ser. SIGIR '11. New York, NY, USA: ACM, 2011, pp. 705–714.

[2] M. Zhang and Z. Zhou, "A review on multi-label learning algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. In Press, 2013.

[3] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998. [Online]. Available: http://research.microsoft.com/\~jplatt/smo.html

[4] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Eleventh Conference on Uncertainty in Artificial Intelligence*. San Mateo: Morgan Kaufmann, 1995, pp. 338–345.

[5] J. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084 – 3104, 2012.

[6] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *J. Mach. Learn. Res.*, vol. 12, pp. 2411–2414, Jul. 2011.

[7] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data mining and knowledge discovery handbook*. Springer, 2010, pp. 667–685.

[8] M. S. Sorower, "A literature survey on algorithms for multi-label learning," Corvallis, OR, Oregon State University, Tech. Rep., 2010.

[9] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, ser. PKDD '01. London, UK, UK: Springer-Verlag, 2001, pp. 42–53.

[10] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038 – 2048, 2007.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[12] P. Malarvizhi and R. V.Pujeri, "Multilabel classification of documents with mapreduce," *International Journal of Engineering and Technology*, vol. 5, pp. 1260–1267, 2013.

[13] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels," in *ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008, pp. 30–44.

[14] C. Mateos, A. Zunino, and M. Hirsch, "Easyfjp: Providing hybrid parallelism as a concern for divide and conquer java applications," *Computer Science and Information Systems*, vol. 10, no. 3, pp. 21–21, 2013.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world Web image database from national university of singapore," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, ser. CIVR '09. New York, NY, USA: ACM, 2009, pp. 48:1–48:9.

[17] R. V. van Nieuwpoort, J. Maassen, G. Wrzesińska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, and H. E. Bal, "Ibis: a flexible and efficient java-based grid programming environment," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 7-8, pp. 1079–1107, 2005. [Online]. Available: http://dx.doi.org/10.1002/cpe.860