Large-Scale Multi-label Ensemble Learning on Spark

Conference Paper · August 2017

DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.328

CITATIONS
2

READS
356

3 authors, including:

Alberto Cano
Virginia Commonwealth University
69 PUBLICATIONS **486** CITATIONS
SEE PROFILE

Sebastian Ventura
University of Cordoba (Spain)
273 PUBLICATIONS **7,831** CITATIONS
SEE PROFILE

Large-scale multi-label ensemble learning on Spark

Jorge Gonzalez-Lopez
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
gonzalezlopej@vcu.edu

Alberto Cano
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
acano@vcu.edu

Sebastian Ventura
Department of Computer Science
University of Cordoba
Cordoba, Spain
sventura@uco.es

***Abstract*—Multi-label learning is a challenging problem which has received growing attention in the research community over the last years. Hence, there is a growing demand of effective and scalable multi-label learning methods for larger datasets both in terms of number of instances and numbers of output labels. The use of ensemble classifiers is a popular approach for improving multi-label model accuracy, especially for datasets with high-dimensional label spaces. However, the increasing computational complexity of the algorithms in such ever-growing high-dimensional label spaces, requires new approaches to manage data effectively and efficiently in distributed computing environments. Spark is a framework based on MapReduce, a distributed programming model that offers a robust paradigm to handle large-scale datasets in a cluster of nodes. This paper focuses on multi-label ensembles and proposes a number of implementations through the use of parallel and distributed computing using Spark. Additionally, five different implementations are proposed and the impact on the performance of the ensemble is analyzed. The experimental study shows the benefits of using distributed implementations over the traditional single-node single-thread execution, in terms of performance over multiple metrics as well as significant speedup tested on 29 benchmark datasets.**

***Keywords*-Multi-label learning; Ensemble learning; Distributed computing; Apache Spark; Big data;**

I. Introduction

Classification is a supervised learning problem in which data is formed by a set of instances associated with a unique class. In traditional classification an instance belongs to a unique class. However, in many real world scenarios one instance may belong simultaneously to multiple classes, which motivates the use of an extended type of learning called *multi-label* classification [1]. In multi-label classification the instances are associated with a set of classes, which are called *labels*. An instance may have multiple labels associated indicating to which categories it belongs.

Multi-label classification has attracted growing interest in the last decade. One of the reasons are real-world applications that fit in the multi-label paradigm: functional genomics [2], recommending bid phrases to advertisers [3], image [4] and music [5] classification among others [6], [7]. Other reasons are new challenges and opportunities introduced by the use of multiple labels, such as the identification and exploitation of dependencies between labels [8], the discretization of the attributes based on the label interdependence [9], and the increased space and time complexity involved in learning from large-scale multi-label data [3].

To address the additional computational complexity of predicting multiple labels, ensemble techniques have become increasingly popular as they have demonstrated the ability to improve the results of individual classifiers [10], [11]. Ensembles are built using a combination of multiple base classifiers. Therefore, their computational complexity increases as the number of base classifiers and the size of the data grows. Eventually, it is not feasible to build ensembles when the number of data instances or labels become unmanageable due to the excessive execution time.

Distributed computing addresses the scalability of ensemble models to larger datasets. The *MapReduce* [12] framework offers a simple and robust paradigm to handle large-scale datasets in a cluster of nodes. This framework is suitable for processing big data because of its fault-tolerant mechanism, which is highly recommendable for long time executions. One of the first implementations of MapReduce was Hadoop [13], yet one of its critical disadvantages is that it processes the data from the distributed file system, which introduces a high latency. On the contrary, Spark [14] provides in-memory computation which results in a big performance improvement in iterative jobs, and hence makes it more suitable for data mining. Spark has been shown to outperform Hadoop by 10x on machine learning jobs [15].

This paper analyzes the characteristics of multi-label ensembles to present five implementations based on parallel computing and Spark. To the best of our knowledge this paper includes the first native implementation of a multi-label ensemble in Spark, without using any wrapper or embedding of traditional classification framework. Specifically, our main contributions are:

- Analysis of the computational complexity of multi-label ensembles.
- Propose five implementations which start from a single-thread and single-node multi-label ensemble, and then scaling to multiple threads and nodes in a cluster.
- Comparative analysis of the predictions of single-node and distributed approaches using classification metrics.
- Extensive experimental study that focuses on the execution time and scalability of the different approaches.

Experimental results indicate that Spark is capable to efficiently distribute the workload, allowing to handle larger datasets. However, it is observed that for small datasets the

2324-9013/17 $31.00 © 2017 IEEE
DOI 10.1109/Trustcom/BigDataSE/ICESS.2017.328

single-node approaches outperform the distributed implementations because the network latency and overheads introduced overcome the actual performance gain. Nevertheless, the use of distributed computing for small data is not justified due to the short runtime.

The paper is structured as follows: Section II reviews related works. Section III describes the characteristics of the ensembles and presents our contributions for improving the ensemble building. Section IV describes the experimental study. Section V shows and discusses the experimental results. Finally, Section VI presents the conclusions of this research.

II. Background

Let's assume a dataset *D*, a single instance (or example) $\mathbf{x} \in D$ is represented as a set of features $\mathbf{x} = (x_1, x_2, ..., x_m)$, where *m* is the number of features. Multi-label considers a finite set of labels $L = \{\lambda_1, \lambda_2, ..., \lambda_k\}$, where *k* is the number of labels. Each of the instances is associated with a subset of labels $L \in 2^k$. These associations are represented as a binary vector $\mathbf{y} = (y_1, y_2, ..., y_k) = \{0, 1\}^k$ where $y_i = 1 \iff \lambda_i \in L$.

A. Multi-label learning

Tsoumakas et al. [1] divide the methods for multi-label classification into two categories: problem transformation and algorithm adaptation.

Problem transformation methods convert multi-label data in multiple representations of single-label transformations, where traditional single-label classification algorithms can be applied. The most important transformations are:

- *Binary relevance (BR)* learns a binary classifier for each different label. The dataset is transformed to *k* datasets D_{λ_i}, *i = 1 ... k*, where *k* is the number of labels. The datasets have only one label λ_i and for each instance will have the value of the original dataset. The main critic to this method is the assumption of label independence, since does not take into account label interdependencies.
- *Label power-set (LP)* transforms the dataset into a multi-class problem, where each class represents an unique combination of labels. The principal advantage of this method is that the set of active labels are modeled together, and as a consequence it maintains the label interdependence. However, the number of classes needed are *min(n,* 2^k*)*, where *n* is the number of instances and *k* the number of labels. This approach becomes unmanageable for datasets with a large number of labels.

Algorithm adaptation methods modify the traditional classifiers to handle multi-label data directly. In [16] they propose the modification of the C4.5 algorithm in order to be able to consider multi-labels in the leaves of the tree. Another adaptation modifies the back-propagation error for neural networks to consider multiple labels [17]. A kernel that supports multi-label data for Support Vector Machines is presented in [18].

A more complex approach is to use a set of multi-label classifiers, which can be either a problem transformation or algorithm adaptation, and combine them to produce a prediction that surpasses the individual ones. The most well known ensembles are:

- *Random k-labelsets (RAkEL)* [19] is a popular ensemble classifier in which each of its components are constructed using a base classifier on a randomly chosen subset of labels. The number of base classifiers by default is the minimum between two times the total number of labels and the binomial coefficient of the number of labels and the size of the subsets. Experiments show that RAkEL improves the results obtained by simply using Binary Relevance and Label Power-set problem transformations.
- *Ensembles of pruned sets (EPS)* [20] is also proposed to address the problem of Label Power-set. The idea is to prune the infrequent sets of labels, and to focus on the most important label correlations with reduced complexity. Therefore, it deletes low represented labels. To construct the pruned sets it samples the data (i.e. bootstrap). The experiments showed that EPS outperformed Label Power-set, and it also proved to be particularly competitive in terms of efficiency, at the cost of discarding underrepresented labels.
- *Hierarchy of multi-label classifiers (HOMER)* [21] is designed for effective and computationally efficient multi-label learning of a large number of labels. HOMER generates a tree of classifiers, the root node considers all the labels and the tree is constructed from top-down. Each of the nodes deals with a smaller set of labels, compared with the total number of labels, and a more balanced example distribution. One of the critical parts is to distribute all the labels into disjoint subsets so that interrelated labels are grouped together.

B. MapReduce, Hadoop, and Spark

The increasing complexity of the algorithms that process large-scale data, in addition to the ever-growing generation of data, require distributed methods to scale data efficiently.

The *MapReduce* framework [12] was developed to process data using a distributed strategy, allowing to scale to data unable to fit in the physical memory of a single machine. This framework provides an abstraction of the underlying hardware and software of the cluster. It partitions, duplicates and distributes the data providing fault-tolerance. Moreover, it schedules the jobs and the network communications, as a result the user only needs to implement the primitives and launch the application.

Hadoop [13] is an open-source implementation of MapReduce. Despite its popularity, Hadoop presents some important weaknesses, such as the impossibility to maintain data in-memory, thus leading to poor performance on iterative methods [22] [23]. On the contrary, *Spark* [14] resolves the previous issue, given that it allows to maintain the data in memory. This allows Spark to outperform Hadoop on iterative machine learning jobs [15].

The main components of the Spark architecture are: The *driver* which loads the application and create a relation of tasks to be executed. The *workers* are the set of different

nodes in the distributed environment, each of them with at least one *executor*. The *executors* are distributed agents that execute the tasks in their local partitions of the data in their assigned memory space. The scheduling of the tasks and assign of the resources to each *executor* is done by the *cluster manager*.

Spark has been successfully applied in different data mining scenarios, such as subgroup discovery [24], real-time fraud detection [25], distributed kNN classification [26], kNN-IS an iterative clustering classifier [27], or entropy minimization discretizer [28], among others.

III. Speeding up ensemble learning

This section presents our contributions for speeding up ensemble learning on large-scale multi-label data. First, it discusses the methodology for computing the base classifiers using parallel and distributed computing. Second, it presents five different implementations based on RAkEL.

A. Building parallel and distributed base classifiers

Traditional implementations build each base classifier iteratively. However, in many ensembles the process of learning base classifiers are totally independent from each other. They do not incur into any data dependencies since they only involve concurrent reads of the dataset. Neither there is task inter-dependencies, since there are no interactions in the learning process among the base classifiers.

There are two approaches to improve the execution times of building base classifiers, which are inclusive. First, each base classifier can be considered independently and built in parallel. Second, the tasks used to build each base classifier can be distributed and later aggregate the results.

In the parallel approach the learning process of each of the base classifiers is performed using parallel threads in a given host. This is straightforward using multiple threads, however is strictly limited by the resources of a single host, in terms of CPU and memory. On the other hand, the distributed approach divides the learning process into smaller tasks that are executed concurrently in multiple machines, taking advantage of both the distributed computation and memory resources, with no latency/overheads due to inter-dependencies. Hence, providing better scalability to handle both bigger ensembles with larger number of components, that need additional computational power, and larger datasets, that need more memory space. The distributed approach using Spark provides transparent parallelism and scalability to the user, as it automatically conducts the remote execution of the jobs in the nodes and the data communications.

B. Parallel and distributed implementation alternatives

This section details the implementations resulted from the combination of the parallel and distributed approaches.

1. Mulan: The first implementation is the original RAkEL from Mulan [29], which is built on top of Weka [30]. The *Mulan* implementation is used as a reference to study the performance of the traditional ensemble methods. This method only supports sequential single-threaded execution, hence it is limited by the resources of a single node. Therefore, it is not scalable to large datasets [31]. The data is read once and loaded into memory, but since the construction of the models is sequential, they do not compete for memory.

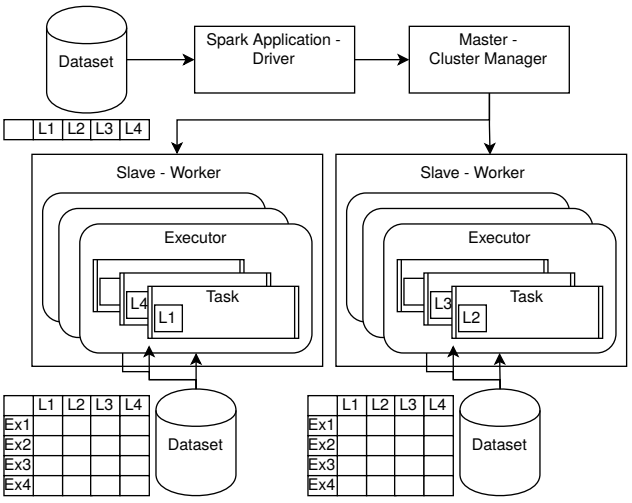

Figure 1: Mulan distributed implementation

2. Mulan threading: This implementation is a modification of the original Mulan. It builds the base classifiers following a parallel approach, as explained in III-A, using multiple threads on the host. It is expected to have a speedup bounded by the number of cores available. However, threads share available memory, limiting its application to large datasets.

3. Mulan distributed: This implementation uses the out of the box classifiers provided by Mulan, and Spark to distribute and schedule the tasks among multiple *executors*. Figure 1 presents how the computation and data are distributed. The *driver* extracts the label information, avoiding reading the whole dataset. It then distributes this information among the *executors*, each having one task which is to build an independent model as if they were a thread in a single node using their own memory space. Every *executor* reads the whole dataset and proceeds to build their models locally. Since the scalability is introduced by the number of *executors* and each of them builds a single-thread model, the best configuration is to have as many *executors* as cores available in each *worker*.

The principal advantage of this approach is delegating the parallelism to Spark. It provides transparency, since Spark allocates the resources. In addition, this approach avoids the communication among *workers*, as well as among *executors*, since they build independent models with their local data. On the other hand, each executor needs to read the entire dataset, resulting in multiple loads of the dataset in the same *worker*.

4. Mulan distributed threading: This implementation is based on the previous one, but uses Spark to control the distribution of the tasks and not the parallelization in each *worker*. Figure 2 illustrates the behavior of this model. As mentioned previously, the *driver* reads the label information and distributes the workload among the *executors*. Each *execu-*

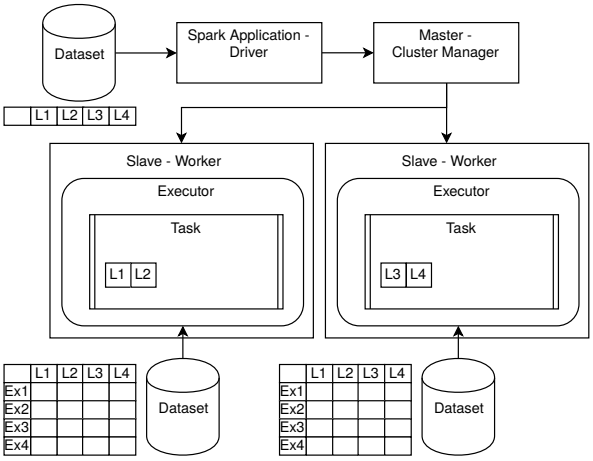

Figure 2: Mulan distributed threading implementation

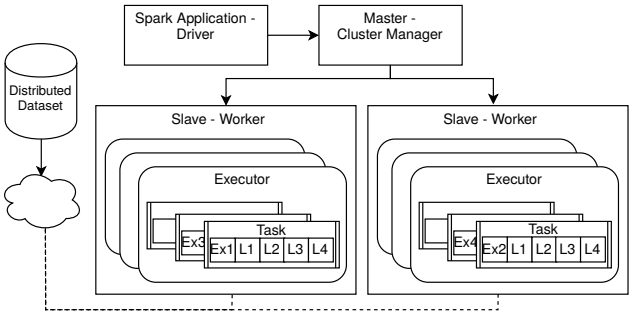

Figure 3: Spark implementation

tor creates multiple threads manually and handles the creation of the models in parallel, thus sharing the memory among the models constructed by a single *executor*. To avoid multiple *executors* competing for the resources in a *worker*, the best configuration is to use a single *executor* per *worker*, which will utilize all the available resources. This implementation handles the construction of the threads manually, avoiding multiple reads of the data per *worker*. However, the same data is still read multiple times in the construction of the ensemble.

5. Spark: The *Spark* native implementation is built on top of the native machine learning library (*MLlib*) [32]. This implementation is illustrated by the diagram in Figure 3. First, the *driver* reads the whole dataset, which is then divided into different partitions and sent to the *executors* through the *cluster manager*. Every *executor* will compute the given tasks in their local partition of data and once completed, send back the results to the *driver*. The construction of each model takes multiple tasks, resulting in an increase network communication between the *workers* and the *driver*.

This approach reads the data from a distributed file system, allowing each *worker* to read the local partitions for its *executors*. Using a distributed file system avoids reading the full dataset, hence allowing to handle large-scala datasets. This approach uses all the available cores, without being dependent of the number of instances or labels of the data.

Table I: Implementation summary

Implementation	Parallel	Distributed	Redundant data reads	Shared memory	Scalability
Mulan	✗	✗	✗	✗	✗
Mulan Threading	✓	✗	✗	✓	# cores
Mulan Distributed	✗	✓	✓	✗	# executors
Mulan Distributed Threading	✓	✓	✓	✓	# cluster cores
Spark	✓	✓	✗	✗	# cluster cores

C. Implementation summary

Table I summarizes the key aspects of the five implementations. It presents the following series of characteristics: parallel construction of the models, distribution of the computation, multiple reads of the dataset, construction of various models in the same memory space, and the resources that influence the scalability.

IV. Experimental setup

This section presents the experimental study carried out to compare the performance impact of each implementation and the parallel and distributed approaches.

Experiments were executed in a cluster with 144 cores and 288 GB of memory. The cluster operating system was Rocks cluster 6.2 x86-64 with Mulan 1.5 for the single-node classifiers and Spark 2.0.0 for the distributed computation.

The predictions obtained by the ensembles were calculated partitioning the data and using the *10-fold cross validation*. The base classifier for the single-node and distributed ensembles is a Binary Relevance transformation with a decision tree whose maximum height is set to eight.

A. Datasets

The experiments were conducted on 29 multi-label datasets. They have been collected from the MULAN[1], MEKA[2] and LABIC[3] repositories websites. Table II list the datasets, which have been compiled attending to a diversity of the two characteristics which are expected to have the most impact in the results: *label dimensionality* and *number of instances*. It also presents the *cardinality* and *density* to measure the imbalance level, a frequent problem for multi-label data [33].

B. Evaluation Measures

The evaluation metrics for multi-label differ from the traditional classification. The most commonly metrics used are proposed in [34] are *Hamming Loss*, *subset-accuracy*, *example-based* metrics and *label-based* metrics. *Hamming Loss* computes the symmetric difference between the predicted set of labels and the true labels. *Subset accuracy* requires that

[1]http://mulan.sf.net
[2]http://meka.sf.net
[3]http://computer.njnu.edu.cn/Lab/LABIC/LABIC_Software.html

Table II: Multi-label datasets and their statistics

Dataset	Inst.	Attr.	Labels	Card.	Dens.
Flags	194	26	7	3.39	0.48
Emotions	593	78	6	1.87	0.31
Birds	645	279	19	1.01	0.05
Genbase	662	1,213	27	1.25	0.05
Medical	978	1,494	45	1.25	0.03
Plant978	978	452	12	1.08	0.09
Enron	1,702	1,054	53	3.38	0.06
Scene	2,407	300	6	1.07	0.18
Yeast	2,417	117	14	4.24	0.30
Human3106	3,106	454	14	1.19	0.08
Rcv1 (subset1)	6,000	601	101	2.88	0.03
Bibtex	7,395	659	159	2.40	0.02
Arts	7,484	526	26	1.65	0.06
Health	9,205	532	32	1.64	0.05
Business	11,214	530	30	1.60	0.05
Social	12,111	539	39	1.28	0.03
Entertainment	12,730	521	21	1.41	0.07
Corel16k	13,766	653	153	2.86	0.02
Society	14,512	527	27	1.67	0.06
Delicious	16,105	1,483	983	19.02	0.02
20NG	19,300	1,026	20	1.03	0.05
EUR-Lex (dc)	19,348	912	412	1.29	0.00
EUR-Lex (ev)	19,348	4,493	3,993	5.31	0.00
EUR-Lex (sm)	19,348	701	201	2.21	0.01
Tmc2007	28,596	522	22	2.16	0.10
Mediamill	43,907	221	101	4.38	0.04
Bookmarks	87,856	708	208	2.03	0.01
IMDB	120,919	1,029	28	2.00	0.07
NUS-WIDE	269,648	209	81	1.87	0.02

the predicted set of labels is exactly the same as the real labels. For both, *example-based* and *label-based* metrics, the basic measures are the same: *precision*, *recall*, *accuracy*, *f-measure* and *specificity*. They just differ in how they are averaged, hence *example-based* gives the same weight to each instance, *micro-averaged label-based* gives more weight to labels with more instances and *macro-averaged label-based* treats all the labels equally reflecting equally less-represented labels [35].

V. Results

This section presents the experimental obtained by the proposed implementations on Section III-B. The results are divided in two groups: prediction metrics and execution times. The analysis of the experiments will first discuss the prediction metrics obtained for the single-node classifiers (Mulan) and the distributed classifiers (Spark), and then it will study the execution times needed to train the ensembles.

A. Evaluation of prediction metrics

In this experiment, we investigate the prediction results of the different implementations on the multiple datasets. To measure the performance of the ensembles we use the multi-label classification metrics presented in Section IV-B which show different perspectives of the same results.

Table III: Prediction metrics averaged for the 29 multi-label datasets and p-values comparison

Type	Metric	Mulan	Spark	p-value
	Hamming Loss	0.0699	0.0670	1.04E-01
	Subset accuracy	0.1298	0.2382	1.70E-05
Example-based	Accuracy	0.2365	0.3717	1.49E-08
	Precision	0.2296	0.4688	1.87E-05
	Recall	0.2695	0.4246	1.49E-08
	F-Measure	0.2746	0.4221	8.80E-06
	Specificity	0.9654	0.9597	8.86E-04
Micro-averaged	Precision	0.5417	0.5721	5.68E-02
	Recall	0.2447	0.3919	1.49E-08
	F-Measure	0.3013	0.4410	1.49E-08
	Specificity	0.9658	0.9603	1.73E-03
Macro-averaged	Precision	0.2928	0.3557	2.09E-04
	Recall	0.2187	0.2941	1.35E-05
	F-Measure	0.2263	0.3001	5.16E-05
	Specificity	0.9560	0.9967	1.20E-04

The Mulan-based implementations all obtain the same prediction results because they learn the same classification model. On the other hand, Spark learns a distributed model, which will produce a different classifier depending on the distribution of the data into the partitions of the nodes.

Table III presents the averaged measures obtained for the different metrics. The measures are grouped attending to the type of averaging: example-based, micro-averaged, and macro-averaged. With the exception of Hamming Loss and subset accuracy, since they do not depend of the averaging process. Table III also presents the p-values, which compares the results obtained by both approaches using the Wilcoxon [36] test. The Wilcoxon rank sum is a non-parametric test recommended by Demšar [37], which allows us to identify whether there are significant differences between the results. The smaller the p-values the bigger the difference between Mulan and Spark.

First, Spark presents more competitive results than Mulan in terms of average values. These differences are produced by the training process of the decision trees. Spark uses the information of the nominal attributes to find better split candidates, thus leading to a considerable improvement of the predictions. This difference is more evident in datasets with a considerable presence of those attributes such as *medical*, *genbase* or *enron*. The results obtained by Spark are not influenced by the number of partitions used, since this parameter only affects to the parallelization level and the number of intermediate operations before aggregating them.

Second, the p-values obtained by comparing the measures are small. The largest values are obtained by the less strict metrics, where small changes in the predictions do not have a big impact on the measure, such as Hamming Loss which evaluates the symmetric difference. This indicates that the results obtained by both approaches are relatively similar. Also, the macro-averaged metrics have larger values compared to the others. These metrics give the same weight to all the labels, producing more balanced results.

However, other type of metrics such as exampled-based or micro-averaged have smaller p-values reflecting larger differences in the measures. Micro-averaged metrics give more weight to the underrepresented labels, hence to the models with bigger differences due to the nature of their instances, and example-based give the same weight per instance producing more diverse results depending on the size of the dataset. This difference can also be appreciated in the averaged measures, and are a result of the specifics of each implementation.

B. Evaluation of execution times

In this experiment, we investigate how the parallelization and distribution of the computation affect execution times in the training process of the multi-label ensembles. To train the ensembles we used the full datasets, without splitting the data into train and test folds. This allowed us to train with all the instances available and study the performance at large-scale.

Table IV compares the implementations using the execution times. The first column shows the execution time on seconds of the original Mulan as the reference. The second group of columns indicate the speedup of the proposed implementations with respect to the original Mulan.

The Mulan threading implementations outperforms the sequential version for every dataset, achieving a linear speedup of roughly four. This implementation has the best results for the smallest datasets. However, the execution times for larger dataset are still unacceptably long.

On the other hand, the performance of the distributed approaches for the Mulan-based and Spark-based implementations is significantly better for larger datasets. The distribution of the data, and hence the computation, comes with a small network overhead due to serialization, transfer and synchronization. This overhead has a significant impact when the data size is small, and therefore it actually takes more time to

Table IV: Execution time of Mulan and speedups of each proposed implementation

Dataset	Mulan - Execution time (s)	Speedup			
		Mulan thread.	Mulan distrib.	Mulan distrib. thread.	Spark
Flags	5.53E-01	5.03	0.12	0.09	0.05
Emotions	2.23E+00	4.09	0.29	0.25	0.16
Birds	9.83E+00	4.14	0.55	0.53	0.35
Genbase	3.94E+00	2.44	0.46	0.35	0.26
Medical	6.71E+01	4.1	4.61	4.01	1.66
Plant978	9.04E+01	4.27	2.53	2.92	2.63
Enron	1.18E+03	3.9	5.15	5.17	19.9
Scene	3.47E+01	4.07	1.52	1.29	2.41
Yeast	3.64E+01	3.84	1.89	1.54	1.19
Human3106	4.40E+02	3.73	6.31	7.08	8.38
Rcv1 (subset1)	1.70E+03	4.83	16.67	18.24	13.47
Bibtex	4.24E+03	4.96	38.97	38.6	26.8
Arts	7.48E+03	4.41	15.53	15.96	100.5
Health	7.80E+03	4.47	15.05	15.13	90.68
Business	1.22E+04	3.81	9.68	12.67	112.37
Social	1.24E+04	3.83	16.87	21.87	131.89
Entertainment	1.21E+04	3.65	10.92	13.89	170.04
Corel16k	8.92E+03	3.71	13.3	13.11	28.08
Society	2.90E+04	3.25	14.67	21.84	369.42
Delicious	1.95E+05	4.59	18.47	19.08	100.91
20NG	4.75E+04	4.05	15.19	31.96	525.04
EUR-Lex (dc)	9.62E+04	4.06	70.46	75.17	111.32
EUR-Lex (ed)	7.89E+05	4.03	96.71	54.88	48.93
EUR-Lex (sm)	3.33E+04	1.97	81.46	77.78	165.5
Tmc2007	2.66E+03	2.94	8.1	12.96	28.83
Mediamill	7.27E+03	3.43	32.72	34.23	40.07
Bookmarks	2.80E+05	4.32	74.04	76.17	513.56
IMDB	1.38E+06	3.73	24.07	27.7	729.07
NUS-WIDE	1.26E+05	4.12	28.47	30.11	437.27

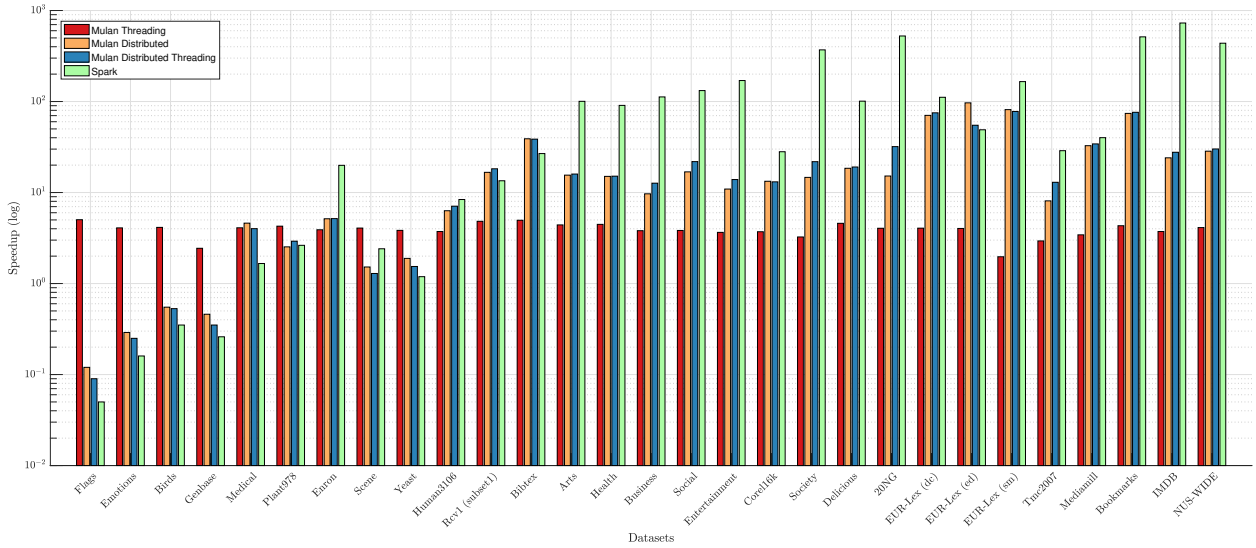

Figure 4: Speedup comparison on each proposed implementation

distribute the data than directly run the computation (achieving speedups less than one). However, this is compensated when the datasets are large enough in terms of instances, which is when the distribution of the data truly makes sense. Now, the bigger the dataset the better speedups achieved.

Interestingly, we noticed that the results for *Mulan distributed* and *Mulan distributed threading* are very similar, with a small advantage towards the threading version. The difference between both methods is how the parallelism in each node of the distributed environment is handled. *Mulan distributed* delegates the parallelization to Spark and *Mulan distributed threading* creates the threads manually, avoiding multiple reads from the same data. The small difference indicates that the multiple reads from the data do not have a big impact over the performance, this is expected since the datasets take less than a few seconds of long execution times. However, this could change in an scenario of millions of instances. Additionally, bigger data takes more space in memory, which eventually can lead to running out of memory sooner in the *Mulan distributed* implementation. Furthermore, this small difference indicate that the overhead introduced by Spark to handle the parallelization is considerably small.

The Spark implementation outperforms the Mulan-based distributed approaches whenever the dataset has at least 7,000 instances (*arts*). This indicates that the overhead introduced to distribute the data among the workers and aggregate the results of the different tasks is only recommended for the large datasets. Again, this implementation achieves the best results for datasets with a large number of instances and/or labels, reducing the time to train the ensembles hundreds of times with respect of the original implementation.

Another important aspect is to consider the evolution of the execution time with regards of the size of the data. Figure 4 presents the speedup of the proposed implementations together for all the datasets sorted by increasing the number of instances.

First, the scalability of the Mulan threading implementation is linear, achieving speedup values limited to the number of cores available in a single node, which is relatively small. Second, the speedup of the distributed implementations scales better the more instances in the dataset. Third, the increase on the number of labels also affects the scalability of the models. Mulan distributed implementations use a single core on the distributed environment to train a given decision tree, which means that when there are more labels than cores in the cluster the behavior will be similar to the Mulan threading implementation. Hence, they are limited by the number of cores in the cluster.

On the other hand, Spark achieves better speedup as soon as the data size grows enough to justify the distribution. Spark distributes the partitioned data and uses all the available cores to train for the partitioned data. This approach is more efficient and allows to handle larger datasets since the limit is set by the memory available, allowing eventually to execute using data from secondary storage. This sets a limit considerably larger than the ones established by the other implementations.

VI. Conclusion

In this paper we have presented and evaluated five alternatives on the scalability of multi-label ensemble classification. RAkEL was selected as a reference model to evaluate benefits of parallel and distributed building of the components of the ensemble. Parallel and distributed approaches have been proposed and compared to the reference classification model in Mulan in order to evaluate classification performance and execution time differences.

The experimental study evaluated and compared the performance of the models with regards to the quality of the results and the execution time considering the data size as measured by the number of instances and labels. The results evaluating the prediction performance indicate that there is not statistical differences between using a single-node and a distributed approach, although in practice Spark produced better results. Regarding the scalability and overall performance, the distributed approaches significantly outperform the single-node version. The native Spark implementation that used the distributed construction of classifiers proved to be the most scalable, maximizing the use of all the available resources in the cluster, especially for large datasets. Spark performed hundreds of times faster than the Mulan implementations.

These results enhance the value of Spark as a solid framework to distribute the workload of large computational tasks, such as multi-label ensembles and presents an open challenge demanding further research.

Acknowledgment

This research was supported by the Spanish Ministry of Economy and Competitiveness project TIN2014-55252-P.

References

[1] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data mining and knowledge discovery handbook*, 2009, pp. 667–685.

[2] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski, "Predicting gene function using hierarchical multi-label decision tree ensembles," *BMC bioinformatics*, vol. 11, no. 1, pp. 2–16, 2010.

[3] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma, "Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages," in *International conference on World Wide Web*, 2013, pp. 13–24.

[4] X. Li, L. Wang, and E. Sung, "Multilabel SVM active learning for image classification," in *International Conference on Image Processing*, vol. 4. IEEE, 2004, pp. 2207–2210.

[5] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas, "Multi-label classification of music into emotions." in *Internation Conference on Music Information Retrieval*, vol. 8, 2008, pp. 325–330.

[6] A. Cano, A. Zafra, and S. Ventura, "A parallel genetic programming algorithm for classification," in *International Conference on Hybrid Artificial Intelligence Systems*, 2011, pp. 172–181.

[7] A. Cano, J. M. Luna, A. Zafra, and S. Ventura, "A Classification Module for Genetic Programming Algorithms in JCLEC," *Journal of Machine Learning Research*, vol. 16, pp. 491–494, 2015.

[8] M.-L. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency," in *International conference on knowledge discovery and data mining*, 2010, pp. 999–1008.

[9] A. Cano, J. M. Luna, E. L. Gibaja, and S. Ventura, "LAIM discretization for multi-label data," *Information Sciences*, vol. 330, pp. 370–384, 2016.

[10] T. Dietterich, "Ensemble methods in machine learning," in *International Workshop on Multiple Classifier Systems*, 2000, pp. 1–15.

[11] M. A. Tahir, J. Kittler, K. Mikolajczyk, and F. Yan, "Improving multilabel classification performance by using ensemble of multi-label classifiers," in *International Workshop on Multiple Classifier Systems*, 2010, pp. 11–21.

[12] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[13] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[14] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Conference on Networked Systems Design and Implementation*, 2012, pp. 1–14.

[16] A. Clare and R. King, "Knowledge discovery in multi-label phenotype data," in *European Conference on Principles of Data Mining and Knowledge Discovery*, 2001, pp. 42–53.

[17] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006.

[18] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in neural information processing systems*, 2001, pp. 681–687.

[19] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 7, pp. 1079–1089, 2011.

[20] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *IEEE International Conference on Data Mining*, 2008, pp. 995–1000.

[21] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels," in *ECML/PKDD Workshop on Mining Multidimensional Data*, 2008, pp. 30–44.

[22] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: a survey," *ACM SIGMOD*, vol. 40, no. 4, pp. 11–20, 2012.

[23] J. Lin, "MapReduce is good enough? if all you have is a hammer, throw away everything that's not a nail!" *Big Data*, vol. 1, no. 1, pp. 28–37, 2013.

[24] F. Padillo, J. Luna, and S. Ventura, "Subgroup discovery on big data: exhaustive methodologies using map-reduce," in *IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 1684–1691.

[25] Y. Dai, J. Yan, X. Tang, H. Zhao, and M. Guo, "Online credit card fraud detection: A hybrid framework with big data technologies," in *IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 1644–1651.

[26] J. Maillo, I. Triguero, and F. Herrera, "A mapreduce-based k-nearest neighbor approach for big data classification," in *IEEE Trustcom/BigDataSE/ISPA*, vol. 2, 2015, pp. 167–172.

[27] J. Maillo, S. Ramrez, I. Triguero, and F. Herrera, "kNN-IS: An iterative spark-based design of the k-nearest neighbors classifier for big data," *Knowledge-Based Systems*, vol. 117, pp. 3 – 15, 2017.

[28] S. Ramírez-Gallego, S. García, H. Mouriño-Talín, and D. Martínez-Rego, "Distributed entropy minimization discretizer for big data analysis under apache spark," in *IEEE Trustcom/BigDataSE/ISPA*, vol. 2, 2015, pp. 33–40.

[29] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2411–2414, 2011.

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

[31] D. Wegener, M. Mock, D. Adranale, and S. Wrobel, "Toolkit-based high-performance data mining of large data on MapReduce clusters," in *IEEE International Conference on Data Mining Workshops*, 2009, pp. 296–301.

[32] X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[33] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006.

[34] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *European Conference on Machine Learning*, 2007, pp. 406–417.

[35] L. Rokach, A. Schclar, and E. Itach, "Ensemble methods for multi-label classification," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7507–7523, 2014.

[36] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[37] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.