Distributed Text Classification With an Ensemble Kernel-Based Learning Approach

Catarina Silva, *Member, IEEE*, Uroš Lotrič, Bernardete Ribeiro, *Member, IEEE*, and Andrej Dobnikar, *Member, IEEE*

***Abstract*—Constructing a single text classifier that excels in any given application is a rather inviable goal. As a result, ensemble systems are becoming an important resource, since they permit the use of simpler classifiers and the integration of different knowledge in the learning process. However, many text-classification ensemble approaches have an extremely high computational burden, which poses limitations in applications in real environments. Moreover, state-of-the-art kernel-based classifiers, such as support vector machines and relevance vector machines, demand large resources when applied to large databases. Therefore, we propose the use of a new systematic distributed ensemble framework to tackle these challenges, based on a generic deployment strategy in a cluster distributed environment. We employ a combination of both task and data decomposition of the text-classification system, based on partitioning, communication, agglomeration, and mapping to define and optimize a graph of dependent tasks. Additionally, the framework includes an ensemble system where we exploit diverse patterns of errors and gain from the synergies between the ensemble classifiers. The ensemble data partitioning strategy used is shown to improve the performance of baseline state-of-the-art kernel-based machines. The experimental results show that the performance of the proposed framework outperforms standard methods both in speed and classification.**

***Index Terms*—Distributed learning, ensembles, kernel-based machines, text classification.**

I. Introduction

In THE last decades, the production of textual documents in digital form has increased exponentially, due to the increased availability of hardware and software [1]. As a consequence, there is an ever-increasing need for automated solutions to organize the huge amount of digital texts produced in applications such as document processing and visualization, Web mining, digital information search, and patent analysis. The task in text classification is often defined as assigning previously defined classes to documents (natural language texts), by analyzing their content. Different methods to choose a classifier can be selected according to experiments on a particular problem instance. However, there are circumstances in which the output of all algorithms solving a particular type of problem are statistically identical [2].

On a particular problem, different algorithms may obtain different results, but for all the problems, they are indistinguishable. It follows that if an algorithm achieves superior results on some problems, it must pay with inferiority on other problems, i.e., the probability distribution on problem instances is such that all problem solvers have similarly distributed results. Consequently, it is of little use to optimize a classifier to a specific problem, since there would be no guarantee of generalization capabilities. Using several classifiers in an ensemble is an appropriate route, when it is possible to take advantage of each classifier benefits avoiding errors, as discussed in [3].

Our approach is to combine different models, inferred with smaller data-driven working sets, in an ensemble of kernel-based machines, namely support vector machines (SVMs) [4] and relevance vector machines (RVMs) [5]. While SVMs implement the principle of structural risk minimization and are widely applied in different applications [6]–[8], RVMs follow a Bayesian approach leading to a probabilistic nonlinear model with a prior on the weights that promotes sparse solutions and are still gaining popularity [9]–[11].

To handle the computational complexity associated with text-classification kernel-based ensemble systems, it is crucial to further include in the framework an underlying distributed nature that permits the deployment in a cluster environment. When processing a computing application in a distributed environment, we often regard the application as a workflow, i.e., a collection of atomic tasks processed in a specific order to accomplish a complicated goal [12]. To define the workflow, we use a directed acyclic graph (DAG) in which the nodes represent individual application tasks and the directed arcs stand for precedence relations between the tasks. Then, we apply partitioning, communication, agglomeration, and mapping strategies to each task definition, and optimize the execution time of the complete classification system.

In this paper, we propose a general framework for text classification with an ensemble distributed approach. The design builds on previous work where ensemble strategies were proposed [13] and simple distributed environments were analyzed [14], [15]. The new framework provides a broad and robust setting for deployment of text-classification tasks.

The rest of this paper is organized as follows. In Section II, we describe the text-classification problem, including

Manuscript received December 29, 2008; revised June 17, 2009 and October 23, 2009. This work was supported by the Ministry of Higher Education, Science and Technology of Slovenia, and the Ministry of Science, Technology and Higher Education of Portugal (2005–2007) under the Slovenia-Portugal Bilateral Scientific Cooperation Project. This paper was recommended by Associate Editor J. A. Keane.

C. Silva is with the Department of Informatics Engineering, School of Technology and Management, Polytechnic Institute of Leiria, Leiria 2411-901, Portugal, and also with the Center of Informatics and Systems, University of Coimbra, Coimbra 3030-290, Portugal (e-mail: catarina@dei.uc.pt).

U. Lotrič and A. Dobnikar are with the Faculty of Computer and Information Science, University of Ljubljana, Ljubljana SI-1001, Slovenia (e-mail: uros.lotric@fri.uni-lj.si; andrej.dobnikar@fri.uni-lj.si).

B. Ribeiro is with the Department of Informatics Engineering, Center of Informatics and Systems, University of Coimbra, Coimbra 3030-290, Portugal (e-mail: bribeiro@dei.uc.pt).

Digital Object Identifier 10.1109/TSMCC.2009.2038280

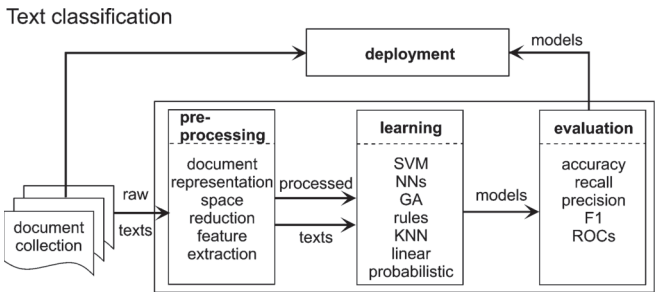

Fig. 1. Text classification: collection, preprocessing, learning, and evaluation.

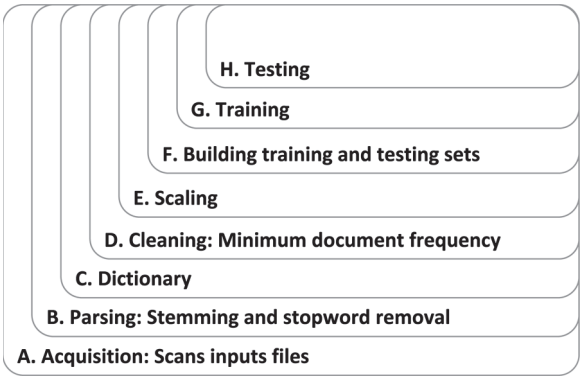

Fig. 2. Text-classification tasks.

representation methods and dimensionality reduction strategies. In Section III, we introduce the learning techniques used in this paper, and in Section IV, we detail the proposed distributed ensemble framework. In Section V, we describe the experimental setup used, including performance metrics and benchmark datasets, and in Section VI, the main results are presented. Finally, Sections VII and VIII address the discussion of results and conclusions, respectively.

II. Text classification

The Internet's phenomenal growth and the widespread use of computers are two major drivers for the development of tools to find, sort, and route information in the most effective and efficient manner. Almost 80% of the information available is stored as text; therefore, the organization of that information has become a complex and vital task [16].

Text classification can be represented by the tasks illustrated in Fig. 1: collection of documents, preprocessing, learning, evaluation, and deployment. In a sequential implementation, each task is carried out by stand-alone executables, capable of processing predefined input files and storing results in corresponding output files. However, when a distributed deployment is intended, dependencies between tasks must be identified. Analyzing a text-classification workflow, we define the general text-classification tasks A to H, represented in Fig. 2, which can later be deployed in a distributed environment.

Task A corresponds to the collection of documents. Tasks B, C, D, and E constitute the preprocessing stage, while tasks F and G comprise the learning phase. Finally, task H matches the evaluation.

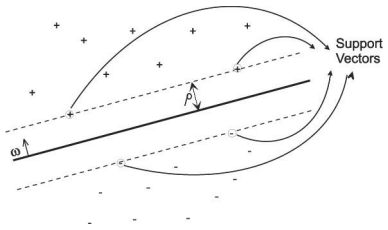

Fig. 3. SVM optimal separating hyperplane.

To represent the documents, we employ the bag-of-words, an attribute value representation used in machine learning [17], where each separate word is a feature (or index term), and its value represents the weight of that feature. Before defining the dictionary to use, space reduction is applied, namely stopword removal, stemming [18], and low-frequency word removal, as in [19]. The representation of terms was the $tfidf$ weight representation system (1)

$$tfidf(w_i) = tf(w_i) \times \log\left(\frac{|D|}{df(w_i)}\right) \tag{1}$$

where w_i is a word or term in the dictionary, $tf(w_i)$ is the term frequency or number of times w_i appears in the document, $|D|$ is the total number of documents, and $df(w_i)$ is the document frequency or number of documents in which the word appears. Before the learning stage, linear scaling (normalization) is carried out to avoid over- or underflow of the learning machines.

III. Learning algorithms

In this section, we first briefly introduce the foundations of the kernel-based machines used in this paper, and then describe the ensemble strategies exploited in the framework.

A. Support Vector Machines

SVMs were introduced by Vapnik [4] based on the structural risk minimization principle, as an alternative to the traditional empirical risk minimization principle. Consider an independent and identically distributed (i.i.d.) sample that consists of N examples, $\{\mathbf{x}_n, t_n\}_{n=1}^{N}$. A general two-class problem is to find a classifier with the decision function $y(\mathbf{x})$, such that $t = y(\mathbf{x})$, where $t = \pm 1$ is the class label for $\mathbf{x}$. The problem can be put as minimizing $(1/2)\boldsymbol{\omega}.\boldsymbol{\omega} + C\sum_{i=1}^{n}\xi_i$ (subjected to constraints), where $\boldsymbol{\omega}$ are the weights of the model, ξ are the slack variables that allow for misclassifications, and C is a parameter that allows the tradeoff between training error and model complexity. Notice that the slack variables account for false-positive and false-negative errors, although in the basic minimization shown, it is not explicit [7]. From the multiple hyperplanes that can separate the training data without error, a linear SVM chooses the one with the largest margin ρ, as shown in Fig. 3. The margin is the distance from the hyperplane to the closest training examples, called support vectors (SVs). The set of SVs also includes the training examples inside the margin and the misclassified ones.

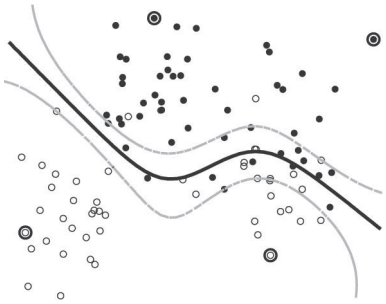

Fig. 4. RVM two-class classification example with four RVs.

B. Relevance Vector Machines

RVMs were proposed by Tipping [5] as a Bayesian treatment of the sparse learning problem for regression and binary classification. The approach shares many of characteristics of SVMs, while avoiding its main limitations, namely, the number of SVs grows linearly with the training set, predictions are not probabilistic, cross-validation to define the parameters is usually needed making the procedure slow, and the kernel function must satisfy Mercer's condition that it must be the continuous symmetric kernel of a positive integral operator [20]. It also typically leads to much sparser models, resulting in faster performance on test data at comparable generalization error. The RVM model equation is given by

$$y(\mathbf{x}, \boldsymbol{\omega}) = \sigma\left(\sum_{m=1}^{M} \omega_m \phi_m(\mathbf{x}) + \omega_0\right) \tag{2}$$

where M is the number of parameters, $\boldsymbol{\omega} = (\omega_0, \ldots, \omega_M)^T$ are the weights, $\phi_m(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_m)$, with $k(\cdot, \cdot)$ being a positive-definite kernel function, and $\mathbf{x}_m$ an example [or a relevance vector (RV)] from the training set. The problem of learning a binary classifier can thus be expressed as that of learning $y(\mathbf{x}_i, \boldsymbol{\omega})$ so that the input feature $\boldsymbol{\phi}(\mathbf{x}_i)$ will map onto the correct classification label $t_i \in \{0, 1\}$. The probability of $\mathbf{x}_i$ to be classified as a class $t_i = 1$ equals $y(\mathbf{x}_i, \boldsymbol{\omega})$, where σ is the logistic sigmoid function.

Under the RVM framework, Gaussian priors are introduced over the weight vector $\boldsymbol{\omega}$, obtaining a model by applying an automatic relevance determination (ARD) prior [21] $p(\boldsymbol{\omega}|\boldsymbol{\alpha}) = \prod_{n=1}^{M} \mathcal{N}(\omega_n|0, \alpha_n^{-1})$, where α_n is the precision hyperparameter of the nth parameter ω_n. Building a Gaussian approximation to the posterior distribution, an approximation to the marginal likelihood can be obtained using Bayes' theorem. Maximization of the approximate marginal likelihood then leads to new values for $\boldsymbol{\alpha}$.

The idea of what an RV is can be gleaned from Fig. 4, which shows a two-dimensional RVM classification example with four RVs. Informally, RVM try to describe the decision surface *as simply as possible* by selecting the RVs as typical instances. On the other hand, SVMs describe the decision surface by selecting the borderline and misclassified instances as SVs [22].

In a batch-learning approach, all training examples will be considered as RVs at the initial stage and the remaining vectors will be pruned after reevaluation of α in each iteration. Since inversion of Hessian matrices is involved in the learning algorithm, the overall training complexity is $O(N^3)$. This implies that if the initial sample size is huge, the learning algorithm may take a long time to converge. Thus, in the baseline RVM experiments, we will use random subsets of 2000 documents.

C. Ensembles

Classifier committees or ensembles are based on the idea that, given a task that requires expert knowledge, k experts may perform better than one, if their individual judgments are appropriately combined. A classifier committee is then characterized by: 1) a choice of k classifiers and 2) a choice of a combination function, usually denominated a voting algorithm. The classifiers should be as independent as possible to guarantee a large number of inductions on the data. Using different classifiers to exploit diverse patterns of errors to make the ensemble better than just the sum (or average) of the parts, we can obtain a gain from synergies between the ensemble classifiers.

1) SVM Ensemble: The purpose of this approach was to develop homogeneous ensembles, i.e., using the same learning algorithm. SVMs are sufficiently scalable to use all training examples for each model (which is not true for RVMs). Thus, we explore different parameters for SVMs [23], resulting in four different learning machines, which are as follows.

1) Linear default kernel.
2) Radial basis function (RBF) kernel.
3) Linear kernel with tradeoff C between training error and margin set to 100.
4) Linear kernel with the cost factor (by which training errors in positive examples outweigh errors in negative examples) set to 2.

Each parameterized model is generated with all the training examples available, and then a majority voting scheme is implemented to combine to outputs, where each base classifier (expert) votes on the class the document should belong to and the majority wins.

2) RVM Ensemble: RVMs have the drawback of low scalability, thus limiting the number of training examples. We have, therefore, devised an RVM ensemble strategy that could take advantage of the overload of training examples in our text-classification benchmarks. The size and number of the training sets used in the RVM ensemble modeling depend on the available computational power, but the more training examples used usually results in more diversity and better performance. In our case, seven smaller evenly sized training sets were constructed, each consisting of 1000 randomly disjointed sample documents[1] from the available training set. Then, from each training set, a model is learned, and these models constitute the ensemble individual classifiers. After this learning phase, and similarly to SVM ensembles, a majority voting scheme is implemented in the testing phase to determine the ensemble output decision, taking as output value the average value of the classifiers that corroborated the majority decision.

[1]We defined a training set of 1000 documents in order to have similar training times for each model, when compared with SVMs.

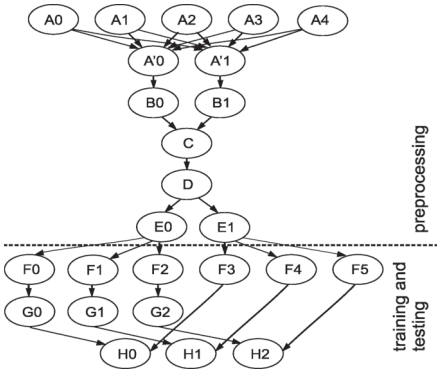

Fig. 5. Example of a DAG representation of a text-classification dataflow.

IV. Proposed framework

To effectively deploy and test the baseline and ensemble kernel-based strategies in a distributed cluster environment, i.e., to define a generic working framework for text classification, we split and organize the text-classification task into subtasks that can be disposed in a DAG (see Section II). Then, we optimize it to run in a definable distributed environment, and we obtain a testable framework.

A. Text-Classification Tasks in a Graph

Each one of the general tasks A–H defined in Fig. 2 can correspond to more than one node in a DAG. For instance, instead of a single task A, we can have five tasks $A0$, $A1$, $A2$, $A3$, and $A4$ that jointly represent the whole task. In this case, an extra task (node), denoted by A', is usually needed to gather (join) the results of the subtasks. A common reason for such partitioning is the possibility of dividing (parallelizing) the inputs into subsets, either because the particular application thus implies or because evident separations exist.

Fig. 5 illustrates how a simple problem can be straightforwardly deployed in a cluster distributed environment. In this case, we considered a toy example with three possible categories and five initial acquisition tasks (tasks $A0$–$A4$ corresponding to task A in Fig. 1). Then, there are two A' tasks to gather results from training and testing examples. Tasks $B0$ and $B1$ correspond to parsing training and testing documents, and tasks C and D are the dictionary building and cleaning (dimension reduction). Tasks $E0$ and $E1$ deal with scaling methods. Finally, task F builds training and testing sets, and tasks G and H correspond to training and testing phases. This initial deployment is usually not optimized. Next section will cover the generic optimization procedure.

B. Optimization of the Graph

DAGs can be efficiently optimized by following the generally adopted methodology of Foster [24], [25]. There are four design steps: partitioning, communication, agglomeration, and mapping. The focus of the first two is to find as much parallelism as possible, while the latter two consider the capabilities of the underlying environment.

In the partitioning step, the data and the computation of a task are divided into small parts that can be computed in parallel. To assure the scalability of the task, divisions where the number of small parts increases with problem size are preferred. When an objective is to use the existing sequential code in parallel design, the data partitioning is more feasible. Otherwise, functional partitioning should be considered as well.

Communication between tasks, which is not needed in sequential designs, represents the overhead of parallel designs. In centralized task scheduling systems, the only possible communication is point-to-point distribution and collection of files at the start and completion of tasks, respectively. In this case, all communication goes through the manager, which can become a bottleneck.

In the agglomeration step, the parts identified in previous steps are grouped into agglomerated tasks in order to improve performance. Grouping should be performed so as to maintain the scalability of the design. Two aspects should be considered to avoid expensive communication. The first is to group the previously identified parts of each tasks in order to optimize communication preferably achieved by sending the same executables and supporting files to each task. The second is to merge consecutive tasks to avoid expensive communication with the master. This is only possible when a certain task needs only intermediary files produced by the preceding task. It is very important to evenly balance the load among computing nodes by creating agglomerated tasks with roughly the same computational and communication complexity.

The mapping step assigns tasks to the computing nodes. When there is a data decomposition in question, the agglomerated tasks usually have very similar complexity, meaning the computational load is balanced among tasks. A good strategy in this case is to create as many agglomerated tasks as there are available processors.

Using the principles in this methodology, a set of general guidelines can now be given to help systematize the procedure, whether it is applied to text-classification applications or not, which are as follows.

1) Time-consuming tasks (bottlenecks) should be identified, parallelized, and distributed (partitioning).
2) The number of subtasks for a task is defined both by an evaluation of the task complexity (building a model) and by the number of available processors (communication and mapping).
3) Partitioning can be accomplished functionally or by data splitting and, depending on the constraints of the specific application, additional processes that join partial results may be needed (partitioning and communication).
4) When dependent tasks exchange large files not needed subsequently, they should be joined (communication and agglomeration);
5) When tasks need few computational resources and have the same input files, they can be joined (agglomeration, communication, and mapping).

The application of this procedure depends on the definition of a model for each task that was carried out by dividing each

task in its fractions: distribution of a task to the computing nodes, the computation itself, and the result collection. The total time needed to complete a set of subtasks can thus be expressed as a sum of these three elements. The distribution burden is dependent on the size of the files to transfer, while the computation depends on the complexity of the task itself. The collection of results depends both on the size of results and on the difficulty of the gathering process. Within this framework, a linear model was proposed in [14]. It was applied to each setting, i.e., to each combination of learning algorithm and dataset to determine the number of nodes to which each task is distributed.

V. Experimental setup

To evaluate the proposed framework, we used a cluster of 16 computers with 3 GHz Pentium 4 processors and 1 GB RAM each, connected over 1 Gb local area network through a fast Ethernet switch. Both Condor and Alchemi middleware platforms for grid computing are set up in the cluster. In the following, before we present and discuss results, the distributed platforms, the experimental collections, and the performance metrics employed are explained.

A. Distributed Environments

The distributed field is booming and offering countless number of platforms and testing possibilities (see [26]–[28]). For text-classification applications, the grid platforms were chosen since they have many advantageous qualities over the classical cluster approaches. First, they do not require major recoding of the existing sequential text-classification applications that we had previously developed for machine-learning purposes. If a cluster approach, such as MPI,[2] was selected, significant effort would have been needed in recoding of existing applications in order to enhance them with the needed functionality. Second, in the classical cluster approaches, the application requirements are usually set at deployment time, leading to noneffective processing with numerous free cycles when the complexity of the application reduces during runtime. In contrary, the grid has ability to dynamically accommodate the number of nodes to the application requirements, i.e., by releasing unneeded nodes or seizing additional resources when necessary. Third, the classical cluster environments consist of usually homogeneous nodes networked in a tightly coupled fashion, while in grid environments, the nodes are often heterogeneous and loosely coupled over existing network connections. Therefore, the grids are better suited for coarse-grained parallel problems among which we can position the distributed applications that try to reuse the existing sequential code. In the experiments, two distributed platforms were selected: Condor and Alchemi. These middleware platforms are adequate for the text-classification tasks that we aim to deploy and are, in fact, two of the most widely used.

1) Condor: The goal of the Condor project[3] is to develop, implement, deploy, and evaluate mechanisms and policies that support high-throughput computing (HTC) on large collections of distributed computing resources. Condor consists of a set of software tools that enable engineers to increase their computing throughput. It is supported for many platforms, including Linux and Windows XP.

2) Alchemi: Alchemi[4] consists of a set of libraries and tools enabling grid computing on the Microsoft .NET platform. It was conceived with the aim of making grid construction and development of grid software as easy as possible without sacrificing flexibility, scalability, reliability, and extensibility. The key features supported by Alchemi are Internet-based clustering of desktop computers without a shared file system, dedicated or nondedicated (voluntary) execution by clusters and individual nodes, and an object-oriented grid thread programming model. The .NET concept with optimized metaassembly codes results in shorter executables resulting in important reduction of communication costs when the distributed grid application is developed from scratch. In case of deploying existing applications in the grid, only the standard .NET communication routines for distribution and collection of executables and data are needed. Since the Alchemi middleware is focused on only one standardized platform, it is likely to be advantageous over a variety of Condor communication protocols used to communicate with many sorts of computer platforms.

B. Experimental Collections

In this section, we give an overview of the text datasets used in the experiments. Two corpora from Reuters were selected, since they constitute widely accepted benchmarks. The datasets have already defined testing splits; therefore, for SVM, all the negative examples were automatically selected. For RVM, the selection of subsets was random, so no real effort was made to differentiate positive from negative examples or to achieve equally balanced training and testing sets. The number of initial features is in the thousands, and after applying stopword removal, stemming, and low-frequency word removal of words that appeared in less than three documents, 497 features were achieved.

1) Reuters-21578: Reuters-21578 is a financial corpus with news articles documents averaging 200 words each. Reuters-21578.[5] In this corpus, 21 578 documents are classified in 118 categories. Reuters is a very heterogeneous corpus, since the number of documents assigned to each category is very variable. There are documents not assigned to any of the categories and documents assigned to more than ten categories. Furthermore, the number of documents assigned to each category is also not balanced. There are categories with only one assigned document and others with thousands of assigned documents. The ModApte split [29] was employed, using 75% of the articles (9603 items) for training and 25% (3299 items) for testing. Table I presents the ten most frequent categories, and the number of positive training and testing examples. These ten categories are widely accepted as a benchmark, since 75% of the documents belong to at least one of them.

[2]Message passing interface

[3]http://www.cs.wisc.edu/condor

[4]http://www.alchemi.net

[5]Reuters-21578 is publicly available at http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html.

TABLE I
NUMBER OF POSITIVE DOCUMENTS FOR THE REUTERS-21578 MOST FREQUENT CATEGORIES

Category	Training	Testing
Earn	2,715	1,044
Acquisitions	1,547	680
Money-fx	496	161
Grain	395	138
Crude	358	176
Trade	346	113
Interest	313	121
Ship	186	89
Wheat	194	66
Corn	164	52

TABLE II
NUMBER OF POSITIVE DOCUMENTS FOR THE RCV1 MOST FREQUENT CATEGORIES

Category	Documents	Training	Testing
CCAT	381,327	23,077	26,923
GCAT	239,267	5,180	44,820
MCAT	204,820	11,110	38,890
C15	151,785	7,454	42,546
ECAT	119,920	7,539	42,461
M14	85,440	4,887	45,113
C151	81,890	698	49,302
C152	73,092	435	49,565
GPOL	56,878	3,756	46,244
M13	53,634	2,613	47,387

2) RCV1: The Reuters Corpus Volume 1 (RCV1) consists of 806 791 English language news documents freely available to the research community.[6] Despite its popularity, Reuters-21578 presents a number of disadvantages, particularly that of overall size (only 21 578 documents). By contrast, RCV1 is 35 times larger. Reuters-21578 covers only a fraction of a year, with somewhat inconsistent coverage of that time period, whereas RCV1 covers a complete year of editorial output, allowing the investigation of temporal issues. In addition, RCV1 was created from a news archive product (i.e., a database) rather than a raw newswire feed, which helps to ensure consistency (fewer duplicates, corrections, brief alerts, etc.). But one advantage of the older collection is that a lot of effort has been put into identifying suitable training/test splits for various applications, particularly those of text categorization and machine learning. In fact, Reuters-21578 constitutes a benchmark for algorithm comparison within the research community, while work on the new corpus is still developing.

Categories for RCV1 are hierarchically organized into four top-level nodes: corporate/industrial (CCAT), economics (ECAT), government/social (GCAT), and markets (MCAT). There are 103 categories currently assigned to the data. In the second column of Table II (documents), the total number of positive documents for the ten most populated classes of RCV1 is given. Besides the four top-level categories, the most populated are GPOL (domestic politics in the GCAT hierarchy), C15 (performance in the CCAT hierarchy), C151 and C152 (accounts/earnings and comment/forecasts in the C15 hierarchy), and M13 and M14 (commodity markets and money markets in the MCAT hierarchy).

[6]RCV1 is available at http://about.reuters.com/researchandstandards/corpus/available.asp.

TABLE III
CONTINGENCY TABLE FOR BINARY CLASSIFICATION

	Class Positive	Class Negative
Assigned Positive	a (True Positives)	b (False Positives)
Assigned Negative	c (False Negatives)	d (True Negatives)

For training, RCV1 defines 22 370 documents that should be used. For testing, we selected the first 50 000 documents not used for training. The third and fourth columns of Table II (training and test) show the effective number of positive documents used for training and testing, respectively. These are the ratios used throughout this paper, and when subsets are used, we choose them randomly.

C. Performance Metrics

In order to evaluate a binary decision task, we first define a contingency matrix representing the possible outcomes of the classification, as shown in Table III.

Several measures have been defined based on this contingency table, such as, error rate $((b+c)/(a+b+c+d))$, recall $((a)/(a+c))$, and precision $((a)/(a+b))$, as well as combined measures, such as, the van Rijsbergen F_β measure [30], which combines recall and precision in a single score, $F_\beta = ((\beta^2+1)P \times R)/(\beta^2 P + R)$. The latter is one of the measures best-suited to text classification [31] used with $\beta = 1$, i.e., F_1, and thus, the results reported in this paper are macroaveraged F_1 values, i.e., performance measures are computed separately for each category and the average of the resulting performance is taken.

VI. EXPERIMENTAL RESULTS

This section presents the results obtained in terms of processing time and classification performance. The proposed framework for distributed text classification is tested on the Reuters-21578 and RCV1 datasets with the SVM and RVM models, as well as with their ensemble versions. Experiments are conducted in both Condor and Alchemi distributed environments, while a centralized approach is used for comparison. We start with initial and optimized DAG structures, and then detail the classification and processing time results.

A. Initial DAG

Using the procedure described in Section IV-A for Reuters-21578 dataset trained with SVMs, the graph in Fig. 6 is easily reached. The acquisition task (A) is divided into 11 subtasks (A0–A10) due to the distributed nature of the used dataset, and these subdivided tasks are, as mentioned in Section IV-A, simply gathered by tasks represented with an apostrophe (A'0–A'3).

Likewise, the parsing task (B) is divided into training and testing sets parsing (B0 and B1), the straightforward functional division, as is scaling (E0 and E1). Building (training and testing) datasets is fully parallelized into 20 subtasks (F0–F9 for training sets and F10–F19 for testing sets). Since there are ten categories, this is a standard procedure. Finally, training and

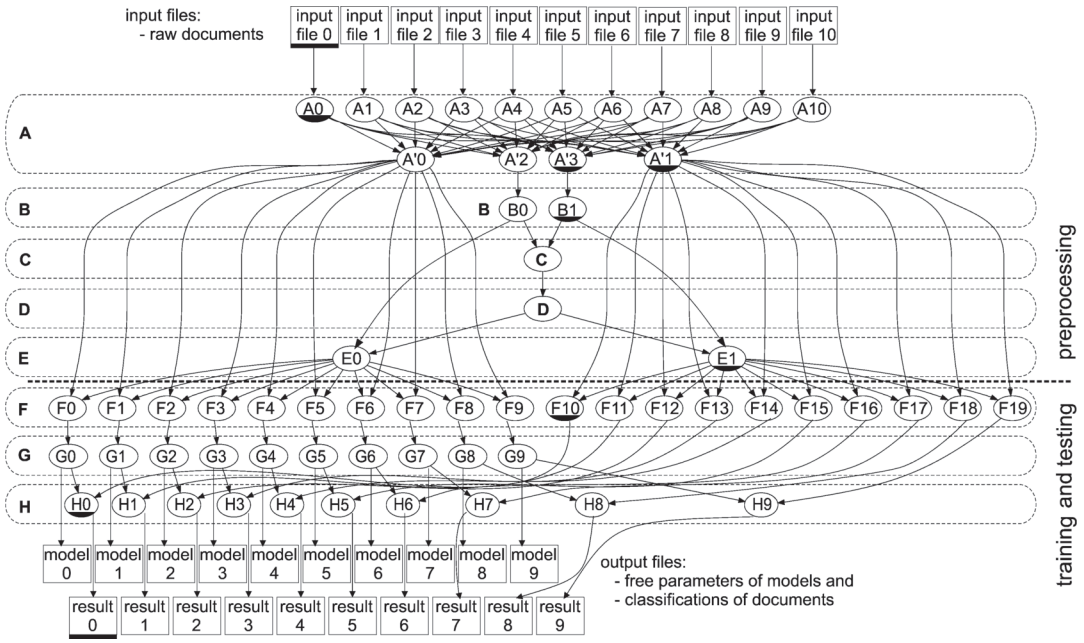

Fig. 6. Text-classification dataflow, represented using a DAG with arrows identifying task dependencies with underlying file exchange. When a new input document is presented to the system and there is no need to update the models, only the files and the subtasks with underlined bottom part are involved.

testing are also divided into one task for each category (G0–G9 for training and H0–H9 for testing).

Also notice that, when a new document is presented to the system and there is no need to update the classification models, only files and subtasks indicated by the underlined bottom part are involved in the processing.

The initial DAGs for the other settings are extremely similar to this one; therefore, we refrain from presenting them. The main changes are in the number of subtasks (essentially in the input and the learning phases). In fact, the construction of this initial DAG is straightforward for any text-classification dataset and inductive learning algorithm.

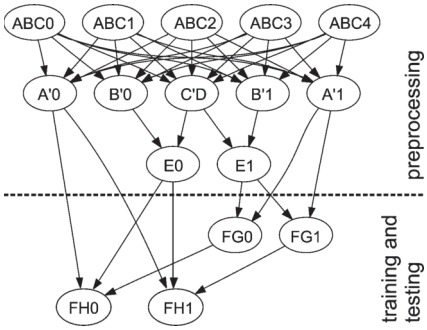

Fig. 7. Text-classification DAG for the SVM model with Reuters-21578.

B. Optimized DAGs

In this section, we will apply the general approach proposed in Section IV-B to the different settings, namely datasets and learning algorithms. We start by fully describing its application to the initial DAG just presented for SVMs, and then point out the differences when applied to the other cases.

Reuters-21578:

1) Tasks A, B, and C exchange many intermediate files not needed in other tasks, and therefore, were agglomerated in an ABC task. To avoid time-consuming experimentation, the suitable number of subtasks (for the main tasks as well as for the gathering tasks) was determined using the model of the environment explained in Section IV-B.
2) Tasks C' and D were joined to combine both partial dictionaries and their document frequencies into the global dictionary and its cleaning. In the latter case, functional agglomeration was performed to reduce file transfer.
3) The task of building training and test sets (F) with SVMs creates large intermediate files for each category, only needed by the subsequent training (G) or testing (H) tasks. For this reason, new joined tasks (FG and FH) were created, reducing the overall file transfer.
4) Similarly to the procedure used with the ABC tasks, the processing times of FG and FH tasks were evaluated to determine the suitable number of subtasks. In both cases, the optimal number of FG and FH subtasks for Condor is two, each one taking care of building or testing five classification models.

As a result of these modifications, we finally reached the refined DAG shown in Fig. 7.

SVM, RCV1:

1) For the tasks ABC and FH the model proposed in [14] suggests ten and five subtasks, respectively, i.e., the optimized balance between computation load of a node and communication overload of partitioning is thus achieved.
2) As the gathered training and test sets are much larger than for the Reuters-21578 dataset, task E was identified as a bottleneck, so it was split into a training task and ten test tasks, as represented in Fig. 8.

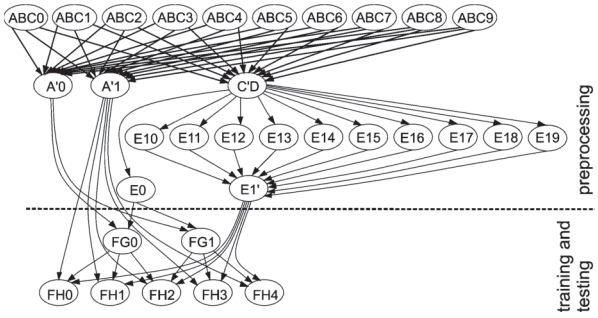

Fig. 8. Text-classification DAG for the SVM model with the RCV1 dataset.

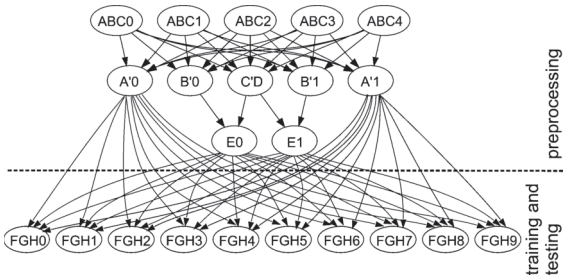

Fig. 9. Text-classification DAG for the RVM model with the Reuters-21578.

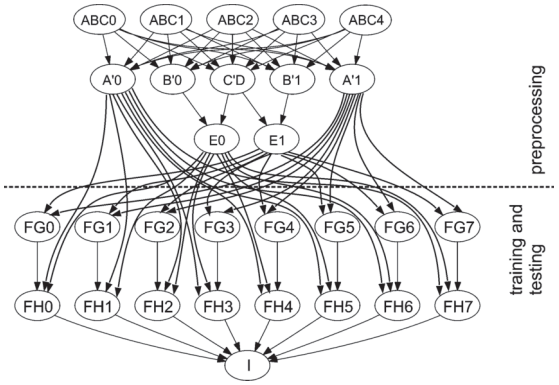

Fig. 10. Text-classification DAG for the ensemble SVM model with the Reuters-21578.

TABLE IV
PROCESSING TIMES USING SVM FOR REUTERS-21578 (IN SECONDS) AND PERCENTAGE OF IMPROVEMENT COMPARED WITH INITIAL VERSIONS (%)

	Sequential	Condor	Alchemi
Initial DAG	516	1487	1075
Optimized DAG	516	543 (-64%)	242 (-78%)

RVM, Both Datasets:

1) Since the datasets are the same as with SVMs, there are no changes in the preprocessing tasks.
2) RVMs learning times are much longer, and therefore, training sets with 2000 training documents were defined.
3) Additional reorganization of training and testing tasks is necessary, since the software package[7] requires the simultaneous input of training and test sets. A joined training and test task, named FGH, was defined.
4) Since the computation largely prevails over communication, it is reasonable to arrange subtasks in such a way that each of them handles training and testing of one of the ten classification models. The refined DAG for Reuters-21578 is given in Fig. 9.
5) In the case of the RCV1 dataset, the training and testing part of the DAG remain unchanged, whereas the initial preprocessing part is the same as the RCV1 with SVMs.

SVM Ensemble, Both Datasets:

1) To adapt the optimized DAG presented in Fig. 7, the learning and evaluation components had to be modified. The training and testing tasks (FG and FH) were thus quadrupled to represent the four different learning machines that constitute the ensemble, as shown in Fig. 10. Moreover, after these tasks, a new dependent task (task I in Fig. 10) was created to implement the ensemble majority voting between the SVM classifiers.

RVM Ensemble, Both Datasets:

1) As with the SVM ensemble, for the RVM ensemble, the changes on the baseline DAG are mainly in the learning and testing tasks (GH). Using the DAG in Fig. 9 as baseline, the RVM ensemble adds an extra task (I) for the ensemble (as in Fig. 10) and 70 FGH nodes (seven for each one of the ten categories in the baseline setting), which makes it difficult to graphically reproduce it here.

[7]http://www.miketipping.com/index.php?page=rvm

C. Processing Times

Each setting presented in this section was tested with 30 runs to establish confidence bounds, and thus, ensure statistical significance using standard deviation. The rationale for this procedure is that, even though the processing times vary slightly in a centralized approach, differences between runs can be significant in distributed environments.

To compare the difference between initial and optimized DAGs, we present in Table IV the processing times for the first DAG emerging directly from the original dataflow (see Fig. 6) and also for the corresponding optimized DAGs for Alchemi and Condor (including the percentage of variation from initial to optimized versions), using SVM models for Reuters-21578.

The initial distributed nonoptimized DAGs are undeniably worse than the sequential approach. This shows how a straightforward deployment in a distributed environment can have adverse effects on processing times, since the distributed text-classification systems approximately double the centralized approach sequential time. Hence, when distributing text-classification tasks (and, in fact, any task), care should be taken to optimize the deployment, as can be realized from the optimized versions results. With some effort of analysis of the distributed environment, an improved solution can be reached, taking full advantage of the computational resources available in the cluster environment. Even for small datasets as the Reuters-21578, gains can be achieved. In this case, Alchemi can reduce by half the processing time (from 516 to 242 s, -53%), while Condor remains competitive (from 516 to 543 s, $+5\%$).

In Table V, we present for each dataset the complete results of the processing times for the optimized DAGs, including the

TABLE V
PROCESSING TIMES FOR THE OPTIMIZED DAGS (IN SECONDS) AND PERCENTAGE OF IMPROVEMENT COMPARED WITH INITIAL VERSIONS (%)

R21578	Sequential	Condor	Alchemi
SVM	516	543 (+5%)	242 (-53%)
RVM	2349	771 (-67%)	735 (-69%)
SVM Ensemble	610	632 (+4%)	472 (-23%)
RVM Ensemble	2960	846 (-71%)	745 (-75%)

RCV1	Sequential	Condor	Alchemi
SVM	5678	2261 (-60%)	1251 (-78%)
RVM	3201	971 (-70%)	874 (-73%)
SVM Ensemble	3700	1690 (-54%)	1171 (-68%)
RVM Ensemble	7365	1486 (-78%)	1423 (-81%)

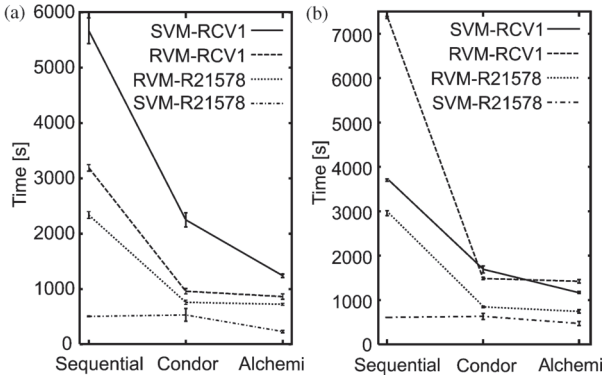

Fig. 11. Final DAG processing times. (a) Baseline algorithms. (b) Ensembles.

percentage of variation. Given the cluster environment, the standard deviations resulting from the 30 experiments were remarkably small, ascertaining the significance of these average results.

The performances are far more expressive for the RCV1 dataset, since it is 35 times larger than the Reuters-21578, which affects both communication and execution. In fact, for Reuters-21578 with SVM, Condor presents in some settings larger processing times. Nonetheless, the distributed approach definitely shows significant improvements, specially for RVMs where the processing times decrease by around three times.

Concerning the ensemble strategy, similar trends to the single learning machine approach are observed. As the learning burden is much larger, the effect of communication and service requirements is less pronounced, leading to even more significant improvement in processing times.

Fig. 11 graphically depicts the processing times for the baseline algorithms [see Fig. 11(a)] and the ensembles [see Fig. 11(b)]. The four settings exhibit the same trend, regardless of the tested dataset. Alchemi always presents a better processing time than Condor, and Condor is generally better than the sequential approach.

D. Classification Performance

In terms of classification performance, baseline SVMs generally provide better average results than RVMs for the same datasets. As can be seen in Table VI (top), the macroaveraged baseline performance presented in the last lines of the table differs by around 10% (from 79.68% to 70.96% for Reuters-21578 and from 61.91% to 49.64% for RCV1). It must be stressed, however, that an RVM uses only a fraction (2000) of the training examples due to the computational constraints, while SVMs use the entire training set. Note also that categories C151 and C152 from RCV1 dataset present worse results than the rest, because they are hierarchically lower in the hierarchy, thus having significantly less positive examples (see Table II).

TABLE VI
R21578 F_1 RESULTS

	Baseline		Ensemble	
Category	SVM	RVM	SVM	RVM
Earn	98.14%	97.52%	97.91%	97.03%
Acquisitions	95.59%	91.57%	95.75%	93.59%
Money-fx	74.64%	57.14%	77.24%	69.62%
Grain	82.76%	76.52%	86.18%	80.30%
Crude	84.87%	69.51%	87.12%	70.86%
Trade	69.90%	50.27%	74.67%	68.57%
Interest	74.73%	58.29%	76.68%	62.92%
Ship	70.59%	66.21%	79.19%	62.77%
Wheat	75.63%	75.18%	83.72%	77.17%
Corn	70.00%	67.37%	79.17%	66.67%
Average	79.68%	70.96%	83.75%	74.95%

RCV1 F_1 RESULTS

	Baseline		Ensemble	
Category	SVM	RVM	SVM	RVM
CCAT	92.78%	82.99%	92.27%	81.55%
GCAT	56.63%	35.83%	62.40%	40.86%
MCAT	90.72%	76.36%	91.40%	77.78%
C15	71.92%	76.47%	78.66%	75.13%
ECAT	74.97%	60.43%	80.11%	65.68%
M14	72.39%	58.72%	78.33%	71.43%
C151	16.01%	31.48%	47.43%	32.95%
C152	5.18%	4.93%	6.13%	3.07%
GPOL	68.78%	57.07%	75.24%	64.37%
M13	69.69%	66.12%	85.52%	75.02%
Average	61.91%	49.64%	69.45%	58.78%

The ensemble strategy generally improves the classification performance comparing with a single learning machine, while maintaining relative performance—around 10% difference from 83.75% to 69.45% for Reuters-21578 and from 69.45% to 58.78% for RCV1, as shown in the last line of Table VI (bottom). The four ensemble SVMs were built by varying parameters using all training samples for each classifier, while for all RVM machines, the same learning parameters and different training samples were used in random subsets of 1000 documents. Considering that SVM and RVM present state-of-the-art results for text classification, their improvement with the ensemble strategy constitutes progress in terms of classification performance.

VII. DISCUSSION

The results presented in the previous section are twofold. On one hand, we have exploited SVMs and RVMs state-of-the-art learning algorithms and expanded them by using ensemble strategies. On the other hand, the major novelty of our paper resides in their distribution in a cluster environment. Both contributions are mutually supporting. The deployment in the distributed environment would not be as providential as was shown, if demanding ensemble methods were not put forward.

Starting to discuss the classification performance (see Table VI), it is evident that SVMs generally perform better than RVMs. This occurs, as noted earlier, due to known scaling problems of the RVMs' that do not allow the use of all training examples. With more training examples, the results would improve. Nevertheless, RVMs maintain a competitive edge and

still provide a number of other nonevident benefits, such as probabilistic outputs.

The ensemble strategy combines diverse classifiers expecting that their error patterns are different and can compensate one another. For instance, when using different classifiers, it is more likely that at any time, only a minority is mistaken, and thus, the ensemble can provide the right classification. Comparing baseline kernel-based machines with ensemble strategies is, as expected, favorable to ensemble strategies, reinforcing the idea that no single classifier is always best. The approaches for RVMs and SVMs were different. In SVM, this improvement is achieved through classifier diversity, a fundamental characteristic of ensembles, while for RVM, the profit is accomplished by subsampling the training set. This procedure also circumvents the computational burden of RVMs, by using a divide and conquer strategy, resulting in a set of prototype RVs that generalize over all the training set. Considering that SVMs and RVMs present state-of-the-art results for text classification, their improvement with the ensemble strategy amounts to a progress in terms of classification performance. For Reuters-21578, the improvement is around 4%, and for RCV1, around 8%, independently of the learning machine.

Concerning the processing times, we will now discuss the settings and results depicted in Fig. 11. These results were achieved for the optimized versions of the DAGs obtained by applying the set of guidelines given in Section IV-B. This procedure is fairly general and can be applied to any constructed initial DAG. In this paper, we used the text-classification workflow presented in Section II, but, in fact, the distribution procedure can be generalized to any DAG, e.g., for different text processing tasks. Additionally, the guidelines are to some extent heuristic, and thus, can result in different optimized DAGs. The proficiency in their application is nevertheless easy to achieve, as demonstrated by the results presented in these case studies.

Establishing a direct comparison between the distributed middleware platforms Condor and Alchemi, a dominance of Alchemi over Condor is noticeable. Condor is a superb tool for high-throughput computing, i.e., heavy computational burdens where the amount of work done is more important than a user waiting for an output. Thus, it is not really equipped with effective interactive tools to deal with highly dependent tasks like the ones defined for text classification. On the other hand, Alchemi is internally much simpler, having less demanding services, and therefore, deals better with task management, which is especially pronounced in frequent handling of tasks when file transfer prevails over the execution burden. Nevertheless, Condor still provides a competitive result, especially when the computational needs are more demanding, viz., RCV1 dataset or ensemble strategies.

In terms of learning algorithms, comparing SVMs and RVMs, the latter exhibit better potential for parallelization, because there is a clear improvement even for the smaller Reuters-12578 dataset. This result was foreseeable since RVMs have noticeably larger computational burdens and were, in fact, parallelized using data partitioning of the inputs. Even so, when the comparison is narrowed to the RCV1 dataset, the improvements for both learning algorithms are comparable and still noteworthy.

One of the most interesting results is in the deployment of ensemble strategies in the cluster distributed environment, as can be gleaned from Fig. 11(b). There is an impressive improvement for the most demanding setting, i.e., RCV1 ensemble, that is evident both for Condor and Alchemi. This result inspires the use of the proposed framework in greater paths, where more ambitious learning techniques can be devised, viz., voting by consensus and boosting.

VIII. Conclusion

In this paper, we have proposed a new systematic distributed ensemble framework to tackle important text-classification challenges, namely, the overload of digital texts and the growth of algorithms' complexity. We described a strategy for efficient deployment of a text-classification workflow in a cluster distributed environment and presented extensive validation of the proposed framework with different case studies. We use two different benchmarks from Reuters, and different learning algorithms and learning strategies.

The framework is based on a kernel-based ensemble system, which uses different baseline kernel classifiers. Specifically, we used SVMs, which, while exhibiting extraordinary generalization capabilities, also scale linearly with the training set. On the other hand, we also made use of RVMs, capable of introducing Bayesian priors which, using *a priori* knowledge, make the model generalize well and incorporate our preferences into the learning algorithm.

The ensemble of kernel-based machines is able to surpass both baseline kernel-based machines by circumventing poor decisions with a strong correct majority of classifiers. One of the appealing factors is that these results were achieved with different approaches to build the ensemble system. While for SVMs, it was constructed using classifier diversity, for RVMs, the ensemble is accomplished by subsampling the training set.

A very important component of this paper is the deployment of the text-classification workflow in a cluster distributed environment. We started by defining a general workflow, easily used in most text-classification tasks, and moreover, easily adaptable to other learning tasks, with different learning algorithms, and even different purposes. We showed how a thoughtless deployment can be inadequate, and proposed a simple and efficient set of guidelines to optimize a DAG. These guidelines constitute a strategy that combines both task and data decomposition, based on partitioning, communication, agglomeration, and mapping to define and optimize the graph of dependent tasks.

Our results relate both to classification performance and processing time, and were achieved using diverse benchmarks. The processing time of a text-classification problem depends on the size of datasets, on the complexity of classification algorithms, and on the deployment in the distributed environment. Processing times taken over 30 runs have shown remarkable improvements over sequential approaches, with little overhead, which was the main purpose of this paper. Moreover, the deployment of ensemble techniques added an improvement in classification performance over state-of-the-art kernel-based machines. We prove experimentally the well-known effect that the parallel processing of the deployed text-classification tasks increases the

performance compared to the sequential processing of the same tasks.

Further research is foreseen in expanding the learning abilities of the classifiers in the distributed environment, particularly by using more complex learning strategies, such as voting by consensus and boosting, now feasible due to the reduction in processing times obtained.

REFERENCES

[1] F. Sebastiani, "Classification of text, automatic," in *The Encyclopedia of Language and Linguistics* vol. 14, K. Brown, Ed., 2nd ed. Amsterdam, The Netherlands: Elsevier, 2006, pp. 457–462.

[2] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[3] L. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. New York: Wiley, 2004.

[4] V. Vapnik, *The Nature of Statistical Learning Theory*. Berlin, Germany: Springer-Verlag, 1998.

[5] M. Tipping, "Sparse Bayesian learning and the relevance vector machine," *JMLR*, vol. 1, pp. 211–214, 2001.

[6] B. Ribeiro, "Support vector machines for quality monitoring in a plastic injection molding process," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 35, no. 3, pp. 401–410, Aug. 2005.

[7] T. Joachims, *Learning to Classify Text using Support Vector Machines*. Norewll, MA: Kluwer, 2002.

[8] M. Nii, S. Ando, Y. Takahashi, A. Uchinuno, and R Sakashita, "Nursing-care freestyle text classification using support vector machines," in *Proc. IEEE Int. Conf. Granular Comput.*, 2007, pp. 665–668.

[9] O. Williams, A. Blake, and R. Cipolla, "Sparse Bayesian learning for efficient visual tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1292–1304, Aug. 2005.

[10] L. Wei, Y. Yang, R. Nishikawa, M. Wernick, and A. Edwards, "Relevance vector machine for automatic detection of clustered microcalcifications," *IEEE Trans. Med. Imag.*, vol. 24, no. 10, pp. 1278–1285, Oct. 2005.

[11] S. Eyheramendy, A. Genkin, W. Ju, D. Lewis, and D Madigan, "Sparse Bayesian classifiers for text categorization," *J. Intell. Community Res.*, 2003.

[12] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 1, pp. 29–43, Jan. 2009.

[13] C. Silva, B. Ribeiro, U. Lotrič, and A. Dobnikar, "Distributed ensemble learning in text classification," in *Proc. Int. Conf. Enterprise Inf. Syst.*, 2008, pp. 420–423.

[14] U. Lotrič, C. Silva, B. Ribeiro, and A. Dobnikar, "Modeling execution times of data mining problems in grid environment," in *Proc. 14th IEEE Int. ERK Conf.,* B. Zajc and A. Trost, Eds., Slovenia: IEEE, 2005, pp. 113–116.

[15] C. Silva and B. Ribeiro, "Speeding-up text classification in a grid computing environment," in *Proc. IEEE Int. Conf. Mach. Learning Appl.*, 2005, pp. 113–116.

[16] M. A. Hearst, "Trends and controversies: Support vector machines," *IEEE Intell. Syst.*, vol. 13, no. 4, pp. 18–28, 1998.

[17] T. Mitchell, *Machine Learning*. New York: McGraw Hill, Jul./Aug. 1997.

[18] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[19] C. Silva and B. Ribeiro, "On text-based mining with active learning and background knowledge using SVM," *J. Soft Comput., A Fusion Found., Methodol. Appl.*, vol. 11, no. 6, pp. 519–530, 2007.

[20] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Piscataway, NJ: IEEE, 1999.

[21] C. Bishop and M. Tipping, "Bayesian regression classification," in *Advances in Learning Theory: Methods, Models and Applications*. (NATO Science Series III: Computer and Systems Sciences). Amsterdam, The Netherlands: IOS Press, 2003.

[22] R. Johansson and P. Nugues, "Sparse Bayesian classification of predicate arguments," in *Proc. CoNLL-2005*, pp. 177–180.

[23] T. Joachims. (2008). SVM light web page. Dept. Comput. Sci., Cornell Univ., Ithaca, NY [Online]. Available: http://svmlight.joachims.org/

[24] I. Foster. (1995). Designing and building parallel programs. Addison-Wesley, Reading, MA [Online]. Available: http://www.mcs.anl.gov/~itf/dbpp/

[25] M. Quinn, *Parallel Programming in C with MPI and OpenMP*. New York: McGraw-Hill, 2003.

[26] I. Foster and C. Kesselman Eds., *The Grid 2: Blueprint for a New Computing Infrastructur*. (The Elsevier Series in Grid Computing). San Mateo, CA: Morgan Kaufmann, 2003.

[27] D. Grosu and U. Kant, "Mercatus: A toolkit for the simulation of market-based resource allocation protocols on grids," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 5, pp. 928–940, Sep. 2007.

[28] N. Liu, M. A. Abdelrahman, and S. Ramaswamy, "A complete multiagent framework for robust and adaptable dynamic job shop scheduling," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 5, pp. 904–916, Sep. 2007.

[29] C. Apté, F. Damerau, and S. Weiss, "Automated learning of decision rules for text categorization," *ACM Trans. Inf. Syst.*, vol. 12, pp. 233–251, 1994.

[30] C. van Rijsbergen, *Information Retrieval*. Newton, MA: Butterworths, 1979.

[31] M. Ruiz and P. Srinivasan, "Automatic text categorization and its application to text retrieval," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 6, pp. 865–879, Nov. 1999.

Catarina Silva (M'10) received the B.Sc. degree in electrical engineering, and the M.Sc. and Ph.D. degrees from the University of Coimbra, Coimbra, Portugal, in 1997, 2000, and 2008, respectively.

She is currently an Adjoint Professor in the Polytechnic Institute of Leiria, Leiria, Portugal. She is also a Researcher in the Centre for Informatics and Systems, University of Coimbra. Her research interest include machine learning and its applications, especially text classification.

Uroš Lotrič received the B.Sc. degree in physics, and the M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Ljubljana, Slovenia, in 1994, 1997, and 2000, respectively.

He is currently an Assistant Professor with the Faculty of Computer and Information Science, University of Ljubljana. His research interests include soft-computing methods, distributed processing, and their applications.

Bernardete Ribeiro (M'01) received the B.Sc. degree in chemical engineering, and the M.Sc. and Ph.D. degrees in informatics engineering from the University of Coimbra, Coimbra, Portugal, in 1975, 1987, and 1995, respectively.

She is currently an Associate Professor in the Department of Informatics, University of Coimbra, where she is also a Researcher in the Centre for Informatics and Systems. Her research interests include machine learning, pattern recognition, text classification, and financial applications.

Andrej Dobnikar (M'94) received the B.Sc. and M.Sc. degrees in electrical engineering, and the Ph.D. degree in computer science from the University of Ljubljana, Ljubljana, Slovenia, in 1971, 1973, and 1984, respectively.

During 1986, he was an Assistant Professor in the University of Ljubljana, where he became an Associate Professor in 1991, and has been a Full Professor since 1996. His research interests include neural networks, soft computing, parallel processing, and adaptive systems.