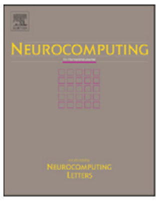
Neurocomputing 108 (2013) 45–57

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



An ontology enhanced parallel SVM for scalable spam filter training

Godwin Caruana [a], Maozhen Li [a,b,*], Yang Liu [a]

[a] School of Engineering and Design, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
[b] The Key Laboratory of Embedded Systems and Service Computing, Ministry of Education, Tongji University, China

ARTICLE INFO

Article history:
Received 23 November 2011
Received in revised form
7 October 2012
Accepted 2 December 2012
Communicated by D. Zhang
Available online 1 January 2013

Keywords:
Spam filtering
Support vector machine
Parallel computing
Classification
MapReduce

ABSTRACT

Spam, under a variety of shapes and forms, continues to inflict increased damage. Varying approaches including Support Vector Machine (SVM) techniques have been proposed for spam filter training and classification. However, SVM training is a computationally intensive process. This paper presents a MapReduce based parallel SVM algorithm for scalable spam filter training. By distributing, processing and optimizing the subsets of the training data across multiple participating computer nodes, the parallel SVM reduces the training time significantly. Ontology semantics are employed to minimize the impact of accuracy degradation when distributing the training data among a number of SVM classifiers. Experimental results show that ontology based augmentation improves the accuracy level of the parallel SVM beyond the original sequential counterpart.

1. Introduction

The overall email ecosystem has become the most ubiquitous modern day communication tool. Its popularity as well as widespread availability have also created an opportunity for a lucrative business model based on unsolicited bulk email, or rather spam. The proliferation of spam has reached widespread proportions. Spam damages enabling communication infrastructures as well as lumbers consumers and service providers in its trail. Such impact and subsequently the importance to identify better ways to control and mitigate its consequences are reflected in the attention that spam continuously gets from various perspectives.

Machine learning techniques such as Support Vector Machines (SVMs) have been applied in spam filtering [1–5]. SVM training is a computationally intensive process primarily due to its convex quadratic programming challenges associated with the dense Hessian Matrix involved during optimization. Numerous SVM formulations, solvers and architectures for improving SVM performance have been explored and proposed [6,7] including the Message Passing Interface (MPI) based parallel approaches [8–10]. SVM decomposition is another technique for improving SVM training performance [11–13]. Less conventional approaches include the utilization of specialized processing hardware such as Graphics Processing Units (GPUs) [14,15]. A widespread practice is to split the training data and use a number of SVMs to process the individual data chunks. This approach typically splits the training data set into a number of smaller fragments. This in turn reduces the individual training set and the overall training time. Most forms of decomposition which are based on a data splitting strategy approach tend to suffer from issues including convergence and accuracy. Challenges related to chunk aliasing, outlier accumulation and data imbalance/distribution tend to intensify problems in a parallel SVM context. This in turn impacts generalization, accuracy and convergence. Techniques such as random sampling have also been shown to exhibit similar accuracy challenges because of the induced probability distribution variance [16]. On the other hand, selective sampling techniques applied to SVMs try to sample the training data intelligently to maximize the performance of SVMs. However these normally require many scans of the entire data set which incurs high overhead in computation.

Consequently, the scalability of SVM in dealing with large data sets still remains a challenge. Our motivation in this work is to research an efficient parallel SVM algorithm based on the highly scalable MapReduce framework [17] for large scale SVM training. SVM training is a global optimization problem which typically relies on the entire dataset to infer the final objective function. Training an SVM by splitting the input data set and working on the individual sub-sets separately may thus reduce the overall accuracy [18]. To this extent, this work also extends our previous effort [19,20] by introducing an ontology based feedback scheme to improve overall accuracy.

The parallel SVM is built on the Sequential Minimal Optimization (SMO) algorithm [11] and implemented using MapReduce.

* Corresponding author at: School of Engineering and Design, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK. Tel.: +44 1895 266748; fax: +44 1895 269782.
E-mail address: Maozhen.Li@brunel.ac.uk (M. Li).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.12.001

Compared with MPI, the MapReduce framework is highly scalable in processing large data sets and it can handle node failures which are critical for long running jobs. In contrast with the sequential SMO algorithm, the parallelized version employs a number of separate SVMs using specific file splits which potentially degrades accuracy in classification. To minimize performance degradation of the parallel SVM, we augment the base training data sets with additional intelligence through an ontology based approach. Ontologies enable the re-use of domain knowledge by ensuring formal and unambiguous concept representation. We transform the SVM testing sets to a Resource Description Framework (RDF) graph representation and identify the misclassified instances using SPARQL [21]. We employ Protege [22] for our prototype and experiments in this context. We re-deploy the augmented intelligence to the original training data sets and re-compute the SVM model. Experimental results indicate that the ontology based approach improves the overall accuracy of the parallel SVM in classification by an average of 4.6%.

The rest of the paper is organized as follows. In Section 2 we briefly discuss some approaches to optimizing SVM in spam filtering. We present a short discussion on the application of semantic and ontology based concepts for tackling spam. In Section 3 we describe the design of the MapReduce based parallel SMO algorithm. We then employ this baseline for accuracy improvement through the application of ontology semantics which is presented in Section 4. Section 5 evaluates the performance of the parallel SMO and compares its efficiency with that of an MPI based approach. We also evaluate the effect of conventional *bagging* and *boosting* techniques on the performance of the parallel SVM. Section 6 concludes the paper and points out some future work.

2. Related work

Spam filtering takes many shapes and forms, and is continuously evolving [23]. MapReduce is employed in a variety of areas ranging from large scale data analysis, search optimization, machine learning, forecasting and social media [24]. The application of *MapReduce* in spam filtering is also common within the industry by the likes of Microsoft, Yahoo, Google [25]. To date, MapReduce primary application is focused towards data intensive tasks rather than for computation in general. However, various efforts to this extent exist as well, including the work presented in [26,17,27].

SVM based techniques have continuously received increasing attention from the research community [2,3,28,29]. The qualities of SVM based classification have been proven remarkable [3,30,10,11,15,28]. The accuracy of SVM can be further improved with numerous methods such as outlier removal, scaling and parameter optimization selection [30]. A SVM kernel typically involves an algorithmic complexity of O (m^2n), where n is the dimension of the input and m represents the number of training elements. Thus, SVM approaches are inclined to be computationally intensive. Scalability of SVM approaches in data training remains an open question due to the involvement of constrained convex quadratic programming challenges. Considerable research has been carried out to this extent to identify ways for performance improvement using varying techniques. Such techniques include optimized SVM kernels, decomposition techniques as well as via the application of distributed computing techniques. SVM approaches rely on the number of support vectors for classification. Kun-Lun et al. [3] try to decrease overall training complexity and classifier categorization by reducing the number of support vectors. Poulet presents an adapted LSVM formulation for SVM training intended to reduce overall complexity [8]. Data chunking, or rather splitting the training data is also applied. This aspect is similar in part to the approach adopted in our work. However, reliance on specialized coding techniques and supporting libraries, rather than adopting widely available and commodity approaches is considered to be a disadvantage. Zanghirati and Zanni [31] propose a decomposition technique for increasing performance and scalability. This takes the form of splitting the overall SVM quadratic programming challenge into a number of smaller sub-problems. Individual results of each separate computation are then subsequently combined. However, the caching strategy of the approach heavily influences the levels of re-computation required for SVM convergence. Cao et al. [32] present another parallel SVM approach using MPI. A comparable approach is adopted by Woodsend and Gondzio [33]. Here optimization is achieved by removing the dense Hessian matrix and matrix partitioning for better data locality. Whilst performance improvement can be achieved by MPI based parallelization, these approaches tend to suffer from poor scalability when heterogeneous computing environments are employed [34,35]. Do et al. [15] present an innovative variation by offloading core processing elements to a GPU (graphics processing unit). The experimental results show remarkable speed improvement when compared with traditional CPU based computation. The application of the GPU is also discussed and featured in the work presented in [14]. Here the authors consider a MapReduce based approach exploiting the multi-threading capabilities of graphics processors. The results show a decrease in processing time requirements in the order of 9 to 35 times. A key challenge with such approaches lies in the specialized environments and configuration requirements. Capitalizing natively on the multi-core capabilities of modern day processors, Chu et al. [36] evaluate the performance of a number of algorithms including a MapReduce based SVM. Dual-core processor design removes most of the communication overhead incurred in distributed processing scenarios. The authors argue that this approach is more pragmatic for real world applications when compared with specialized implementations such as the one presented in [6] in which a cascaded SVM scheme is presented. Each training data set is used as input for each SVM instance and an iterative re-training (cascading) process is applied. The effectiveness of such an approach relies primarily on the number of cascade iterations performed. Where substantial iterations are required, the overall effectiveness can be reduced because the time required to combine the respective support vector sets cannot be discounted. This issue is also highlighted in [35].

The application of ontology and semantics in the context of spam filtering facilitates the definition and understanding of spam in a better and more formal way. The ability to exchange intelligence and subsequently the potential for machines to process it in a formal and interoperable fashion provides numerous opportunities. Annotating email messages with metadata brings numerous benefits including augmented intelligence, context richness and formalization. The incorporation of domain knowledge can facilitate automated filtering processes as well as increase classification accuracy. Kim et al. [37] propose an adaptive ontology for email filtering using a J48 decision tree based approach. This approach is built on a pre-trained Weka [38] model which is subsequently translated into an RDF based ontology representation. Jena is employed for generation of the actual ontology from the Weka decision tree model. The ontology representation generated by Jena subsequently provides a number of assertions which are employed to classify email as spam or ham. Ontological knowledge can also be built by identifying and formalizing the relationship between user's choices as well as how spam is reacted to (i.e., replied to). This approach forms the basis behind the work presented in [39] where the authors classify 'reaction' to email into four broad categories, namely

'Reply', 'Delete', 'Store' and 'Spam'. On this basis, association mining is applied to an initial reference data set using Weka. This approach influences the classification by differentiating by the action performed as well. Balakumar and Vaidehi [40] also consider the challenge of spam from a personal dimension—an email which is spam to someone may be considered and treated as legitimate by someone else. Here, a combination of Bayesian and ontology approach is considered for email classification. The key advantage of the proposed approach is the inherent simplicity to integrate with current approaches. The challenge however lies where different personalized filters exist for a large user base. Scalability has become a challenging prospect where the complexity and size of individual personalized filtering increases significantly. Also, concept drift is an acknowledged challenge in model learning tasks [41]. Han et al. [42] consider the notion of concept drift in spam filtering for adaptive learning. They argue that users may change their opinion with respect to interest and subsequently classification of certain mail types. To increase accuracy, the authors analyze end user's intent with respect to actions undertaken. Other approaches for tackling specific types of spam such as that based on images which use ontology as an enhancing technique also exist [43–47].

3. Parallelizing SVM with MapReduce

MapReduce is a parallel and distributed programming model in support of data intensive applications. Programmatically inspired from functional programming, at its core there are two primary features, namely a *map* and a *reduce* operation. From a logical perspective, all data is treated as a Key (K), Value (V) pair. Multiple *mappers* and *reducers* can be employed. At an atomic level however a *map* operation takes a {K1, V1} pair and emits an intermediate list of {K2, V2} pairs. A *reduce* operation takes all values represented by the same key in the intermediate list and generates a final list. Whilst the execution of *reduce* operations cannot start before the respective *map* counterparts are finished, all *map* and *reduce* operations run independently in parallel. Each *map* function executes in parallel emitting respective values from associated input. Similarly, each *reducer* processes different keys independently and concurrently. An abstract representation of a typical MapReduce framework is illustrated in Fig. 1.

As indicated earlier, SVM training is a computationally intensive machine learning dimension for spam classification. In the sequential SMO implementation, most of the computation stems from the iterative kernel evaluations to update the optimality condition vector [35] (Eq. (1)).

$$\sum_{j=1}^{l} a_j y_j K(X_j, X_i) - y_i \tag{1}$$

A typical approach to parallelizing this operation is presented in [14] based on which we parallelized SVM with MapReduce. Similar to the work presented in [32], the computation of the objective function (Eq. (1)) in our work is performed as a Map operation in the context of MapReduce. However, the parallelization of bias computation in [32] is performed in such a way that each optimization iteration requires a Map operation as well as a Reduce operation. This approach introduces a substantial network communication overhead due to the large number of iterations to be performed. In our approach, a single MapReduce step is employed for each data fragment (partition) which reduces the overall computation and network communication overhead significantly.

The implementations of MapReduce include Mars [48], Phoenix [49], Hadoop [50] and Google's implementation [51]. Hadoop's high performance (for example, during 2008, a 900+ Hadoop cluster sorted 1 Terabyte of data in approximately 200 s [52,53]), user base and support community, flexibility and open source nature were the primary reasons for its adoption within this research work. From a MapReduce perspective and in the context of Hadoop, the data splitting strategy can be done according to the number of MapReduce tasks that will be employed. Each Map task ($MAP_1 \ldots MAP_n$) will process the associated data chunk ($DataChunk_1 \ldots DataChunk_n$) and generate a respective set of Support Vectors ($SV_1^{set} \ldots SV_n^{set}$). In this particular scenario, these are then forwarded to a single Reducer ($REDUCE_1$) which will contribute the, respectively aggregated Support Vector

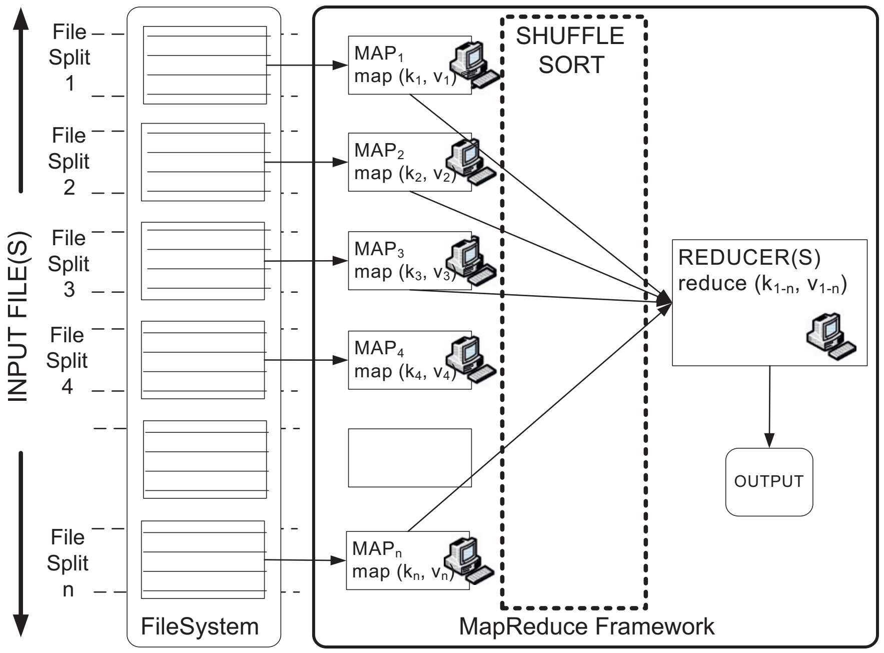


Fig. 1. MapReduce framework.

Set (SV), weight (w) and bias (b) elements of the global SVM to a final learned model. Given that this is a linear SVM model [54], in our prototype, the aggregation of the weight (w) and bias (b) elements are performed using a sum and average strategy, respectively. In a linear SVM, an optimal hyperplane is the one with the maximum margin of separation between the two classes, where the margin is the sum of the distances from the hyperplane to the closest data points of each of the two classes.

SVMs have been shown to be adaptable to the Kearns statistical query model and associated summation form, and thus fit well into the MapReduce framework [36,55]. Adopting a sum/average strategy for establishing the global weight vector and bias is both computationally light from an algorithmic perspective and recognized accurate [36]. From a MapReduce perspective, this strategy also allows us to perform aggregation via a single Reducer, decreasing MapReduce framework overhead. The final output will be used as the final classification model including the necessary information for the objective function to be able to classify unseen data. A high level pictorial representation of this approach is shown in Fig. 2.

Algorithm 1 presents the parallel SMO based on the sequential one described in [11]. The *map* segment of the algorithm is basically the same as the original SMO algorithm except that it is applied for each participating *mapper*. The primary difference lies in the ways that the global *b* threshold (b_{global}) and weight vector (w_{global}) are computed via the *reducer*, using an *average* and *sum* strategy, respectively as described in the pseudo code.

Algorithm 1. MapReduce based parallel SMO.
where:

- Map_j=MapReduce Map
- Data_{chunk}=training data associated with Map_j
- x=training elements, y=class labels for x
- w_j=local (Map_j) weight vector
- b_j=local (Map_j) b threshold
- I=training data index set
- α_j=Lagrange multiplier(s)
- b_{global}=global b threshold
- w_{global}=global weight vector

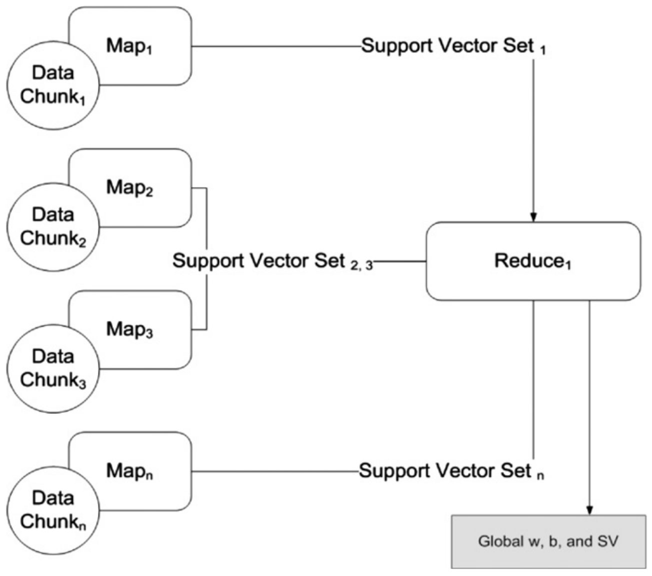


Fig. 2. Aggregation of SVM process.

Algorithm 2. Map Reduce SMO Algorithm (linear SVM).

```
1  MAP_j ∀_j ∈ {1…data_chunk}
2  input: set of training data $x^i$, corresponding labels $y^i$, $\forall i \in \{1\ldots l\}$
3  output: weight vector $w_j$, $a_j$ array, $b_j$ and SV
4  initialize: $a_i \leftarrow 0$, $f_i \leftarrow -y_i$ $\forall i \in \{1\ldots l\}$
5  compute $b_{high}$, $I_{high}$ , $b_{low}$, $I_{low}$
6  update $^aI_{high}$ and $^aI_{low}$
7  repeat
8      update $f_i$, $\forall i \in \{1\ldots l\}$
9      compute $b_{high}$, $I_{high}$, $b_{low}$, $I_{low}$
10     update $^aI_{high}$ and $^aI_{low}$
11 until $b_{low} \leq b_{high}, +2\Gamma$
12 update $b_j$ bias term
13 store updated $a_{j1}$ and $a_{j2}$
14 update $w_j$
15 REDUCE
16 input : set of Map_j weight vectors $u_j \forall_j \in \{1\ldots data_{chunk}\}$ ,
       set of Map_j bias $b_j$ $\forall_j \in \{1\ldots data_{chunk}\}$
17 output : global weight vector $u$, average b and SV
18     compute $b_{\text{global}} = \sum_{j=1}^{Map_j} b^{data}{}_{chunk} / Map_j$
19     compute $u_{\text{global}} = \sum_{j=1}^{Map_j} w^{data}{}_{chunk}$
```

In Algorithm 1, for each Data_{chunk} we associate a Map (Map_j) operation. In this context, line 4 initializes the necessary structures, primarily the α multipliers and the objective function. Lines 5–10 portray the SMO optimization process. Iterations are based on the selection and optimization of two Lagrange multipliers, subsequently the objective function. Line 11 checks for the respective exit conditions, whilst line 12 updates the bias threshold accordingly. For actual implementations, multiplier selection is frequently based on approaches such as heuristics, albeit strategies vary with specific implementations. Lines 13 and 14 store the Lagrange multipliers and update the local weight vector for the specific *map* (Map_j).

In contrast with the sequential SMO algorithm presented in [11], we perform two additional steps using the *reduce* phase of the MapReduce prototype. Basically, the *reducer* takes the following steps:

- Performs an average computation on all respective *b* outputs emitted by the individual Map (Map_j) operations $=> b_{\text{global}}$ (Line 18)
- Performs a sum operation on the weight vectors emitted by the respective Map (Map_j) operations $=> w_{\text{global}}$ (Line 19)
- Performs the above two steps to generate the global *b* and weight vector for subsequent classification.

We employed Weka's SMO [56] implementation as a baseline solver. For this work we focused on linear SVMs, although the approach can be easily extended and applied to non-linear variants as well. The base SMO algorithm is decomposed and re-structured to benefit from MapReduce. Each MapReduce *map* processes an associated data chunk in its entirety. The output of each *map* process is the localized (per data chunk) SVM weight vector (Algorithm 1: w_j) and the bias (Algorithm 1: b_j) threshold. The primary role of the associated *reduce* phase is to compute the global weight vector (Algorithm 1: w_{global}) by summing the individual *maps* weight vectors. The bias thresholds from each *map* output are averaged by the respective *reduce* phase (Algorithm 1: b_{global}). More formally, each individual Map_j is the partial weight vector and the value of *b* for the respective

Data$_{chunk}$ partition (Eq. (2)):

$$\underset{partial}{\vec{w}} = \sum_{i=1}^{l} y_i a_i \vec{\chi}_i \tag{2}$$

The global weight vector is computed via the reducer by summing all partial weight vectors from each respective mapper (Eq. (3)) as well as averaging all b values for all Data$_{chunk}$ (Eq. (4)). The value of b is also handled by the reducer which averages the value of b across the partitions/mappers. To calculate the SVM output, the global weight vector and the value b are required (Eq. (5))

$$\vec{w}_{global} = \sum_{i=1}^{n} \vec{w}_{partial} \tag{3}$$

$$\vec{b} = \frac{1}{n}\sum_{i=1}^{n} b_{i\ partial} \tag{4}$$

$$u = \vec{w} \times \vec{x} - b \tag{5}$$

For a non-linear SVM [57], each mapper's output would be the alpha array for the respective local partition and the value of b. The reducer would join the partial alpha arrays to produce the global alpha array. As with the linear SVM approach, the reducer deals with the value of b averaging its value of across partitions.

In this respect, from a time complexity perspective the original single sequential representation, i.e., $O(m^2 n)$ can now be contextualized in a MapReduce environment and expressed as (Eq. (6)):

$$O((m^2 n/s) + n\log(S)) \tag{6}$$

where:

n=the dimension of the input
m=the training samples
S=the number of MapReduce nodes

4. Ontology augmentation

In order to improve the overall classification accuracy of the parallel SMO algorithm, we extend it with an ontology based enhancement process. We have designed *SPONTO*, a spam ontology which acts as a feedback loop base for the training and classification processes. This is in contrast with the work presented in [39] where the ontology itself is employed for classification. The feedback loop is then employed to re-train the parallel SMO with added intelligence to improve overall accuracy. This is performed to mitigate the accuracy degradation challenge introduced due to the training data file splitting strategy and respective separate SMO computation. SPONTO reflects all the basic elements presented in the SpamBase [58] dataset as well as additional attribute assertions. SpamBase is a multivariate dataset containing 4601 mail instances and formulated as a series of 58 attributes or feature representations. These are mostly derived from character or word frequencies that describe the original mail message content. Around 40% of the dataset is spam. The last identifier of the dataset is the *nominal class label* which designates whether the email represents ham or otherwise. SPONTO is employed to provide users with additional intelligence in terms of mail class (Ham or Spam), the classification result of the parallel SMO as well as support for instance weights. From a very high level perspective the core basic SPONTO structure can be described as follows:

- A base root class designated EMAIL
- Two key classes, SPAM and HAM, sub-classes of EMAIL
- SpamBase instance attributes, represented as data property assertions within SPONTO
- Supplementary instance attributes reflected as additional property assertions indicating whether instances have been misclassified via the SVM classifier, instance weights and mail class types.

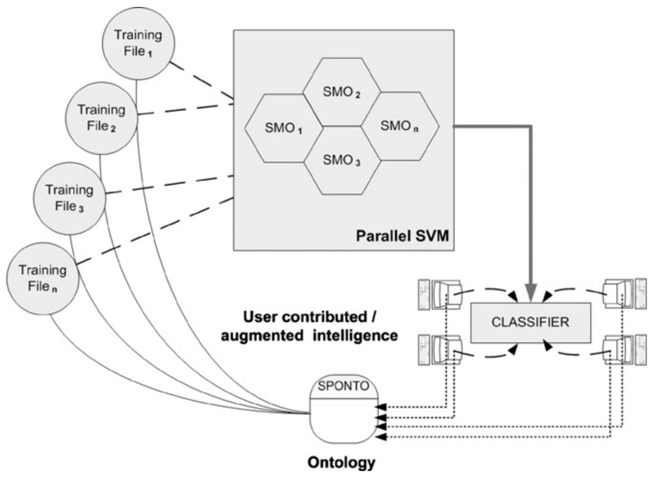


Fig. 3. Ontology assisted classification process.

The intelligence conveyed through the supplementary instance attributes via end user contribution is employed for correcting and influencing training data. The end user contributed ontology intelligence augmented training sets are employed for the regeneration of the classifier by the parallel SVM as shown in Fig. 3.

The key steps of the ontology enhanced parallel SVM process are as follows:

- Employ split data training sets with the parallel SVM to compute and output the SVM classifier.
- Use the SVM classifier to classify Ham/Spam testing set and create respective SPONTO instances.
- Using ontology (SPONTO), user corrects ontology instances (SPONTO-I).
- Merge user modified instances (SPONTO-I) as well as corrected classifications and update the training data for the parallel SVM for further processing.

More specifically, a base RDF graph from the SpamBase ARFF file is generated based on the SPONTO ontology structure. We transform the Weka ARFF format to an equivalent RDF representation as portrayed in the simple example presented in Table 1. We then apply the learned model from the parallel SVM on the instances which require classification. For each testing instance, we generate a new ontology instance based on SPONTO. Ontology generation can take two different paths:

- The first is via the extraction of instance data and generation of a respective SPARQL query. This query is applied on the base RDF graph to identify respective misclassified elements.
- The other approach is performed by applying an intermediary J48 classifier. In this approach, fact rules generated by the C4.5 decision tree algorithm are transformed into respective SPARQL queries. Once again, the respective queries are executed on the base RDF graph to identify misclassified elements.

In either approach, misclassified nodes are identified as a set of final ontology instances. The misclassified instances from the ontology based representation are automatically correlated, through SPARQL [21], with the original training instances and the latter presented to the end-user. Users can subsequently contribute preference and intelligence by increasing individual

Table 1
Weka to RDF conversion.

```
WEKA
@relation spambase
@attribute word_freq_make     numeric
. . . . . .
@attribute class {1,0}
@data
0,0,0,0,1.82,0.36,0.36,0.72,0.36,0.36,0,0,0,0,0,0,0,0.36,2.91,0,2.18,0,0.72,0,0,. . . . . . . . . . . . . . . . . . . . . . .
.,0.297,0.059,0.178,0,0,2.446,11,115,1
. . . . . .
RDF
<?xml version="1.0"?>
<rdf:RDF xmlns:mail="http://localhost/sponto/mail#" xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema3">
    <rdf:Description rdf:about="http://localhost/sponto/mail#word_freq_make">
        <rdf:type>      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://localhost/sponto/mail#Email"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
        </rdfs:range>
    </rdf:Description>
. . . . . .
```

training instance weights, removing instances or modifying instance classification outcomes. For this work, the actual influencing step is therefore performed manually, but we intend to change this for future work as indicated in Section 6. For the proposed prototype, we employ Protege, the Ontology Editor and Knowledge Acquisition System [22] for Ontology interaction and end user contribution. Ontology reasoners can also be employed to explore and validate the generated ontology base—available as respective Protégé Plug-ins in this particular case and which include HERMIT, PELLET and FACT amongst others. This provides the ability to perform inference and assertion operations, identify inconsistent concepts and equivalency on the ontology. Additional interaction with the ontology, including extensive querying such as via Manchester Syntax [59] based DL-Query language can also be performed to fine tune ontology instance quality. The final process involves the re-generation of the Weka ARFF input files for subsequent processing by the parallel SVM from the final ontology. We increase correctly classified instance weights, correspondingly decrease the instance weights of incorrectly classified ones and merge these instances with the original input source. Training instance weighting also allows us to perform a degree of noise mitigation in a simple yet effective way. In fact, instance weighting has a number of advantages when compared to discarding [60]. The overall classification process is portrayed in Fig. 4.

5. Experimental results

A number of initial experiments were carried out to identify the accuracy and efficiency of the parallel SMO when compared with the sequential counterpart. For all classification experiments carried out, the SpamBase [58] dataset was employed. There are 4601 instances in the original SpamBase dataset. A baseline experiment intended to identify typical sequential SMO performance using the SpamBase dataset on a typical desktop machine was performed. The configuration of the desktop machine is shown in Table 2.

Weka's default SMO parameters were employed, namely c (complexity) set to 1.0, epsilon (round-off error epsilon) set to 1.0E-12, Polynomial Kernel. The original dataset was split into a number of smaller sub-sets and concatenated to create larger input data sets where required. Weka's SMO classification scheme was employed [56], using a number of unlabeled instances and varying the number of training instances. Fig. 5 shows the processing times of the sequential SMO algorithm during training. From Fig. 5 we can observe that the number of training instances varied from 204 and 128,000 with the training time ranging from 1 s to 563 s. It is worth noting that the sequential test failed when a training attempt with 327,750 elements was performed. Based on the 4601 instances of the SpamBase dataset, using the Weka random re-sampling filter feature, a varying training input size ranging from 204 and 128,000 training instances was employed. The sequential test failed when an attempt with 327,750 elements was performed. Respective accuracy ranged from a minimum of 81.04% correct to a maximum of 94.03% correct as shown in Fig. 6.

5.1. The efficiency of the parallel SMO

In a second experiment, the approach taken for the sequential SMO was re-modeled for testing on a MapReduce Hadoop cluster. The SMO algorithm provided in Weka was extended, configured and packaged as a basic MapReduce job. The Hadoop cluster was configured with the resources as shown in Table 3.

The time required to train the SMO sequentially using 128,000 instances on a single computer node was $\approx$563 s whilst the parallel SMO consumed $\approx$134 s using the Hadoop cluster with 4 computer nodes. The sequential SMO for testing 327,750 instances failed because of the large number of instances. Fig. 7 compares the efficiency of the sequential SMO in training with that of the parallel SMO using a varying number of nodes.

5.2. A comparison with an MPI based SMO

To further assess the performance of the MapReduce based SVM, we compared it with an MPI based parallel SMO algorithm presented in [32]. The maximum speedup recorded via the MPI approach is 21 times using a 32 processor configuration. We evaluated the MapReduce SVM using the same Adult dataset [61] adopted in the MPI work. This specific data set, containing 48,000 instances, predicts whether income exceeds $50,000 per year based on census data. Two sets of tests, namely using a standard

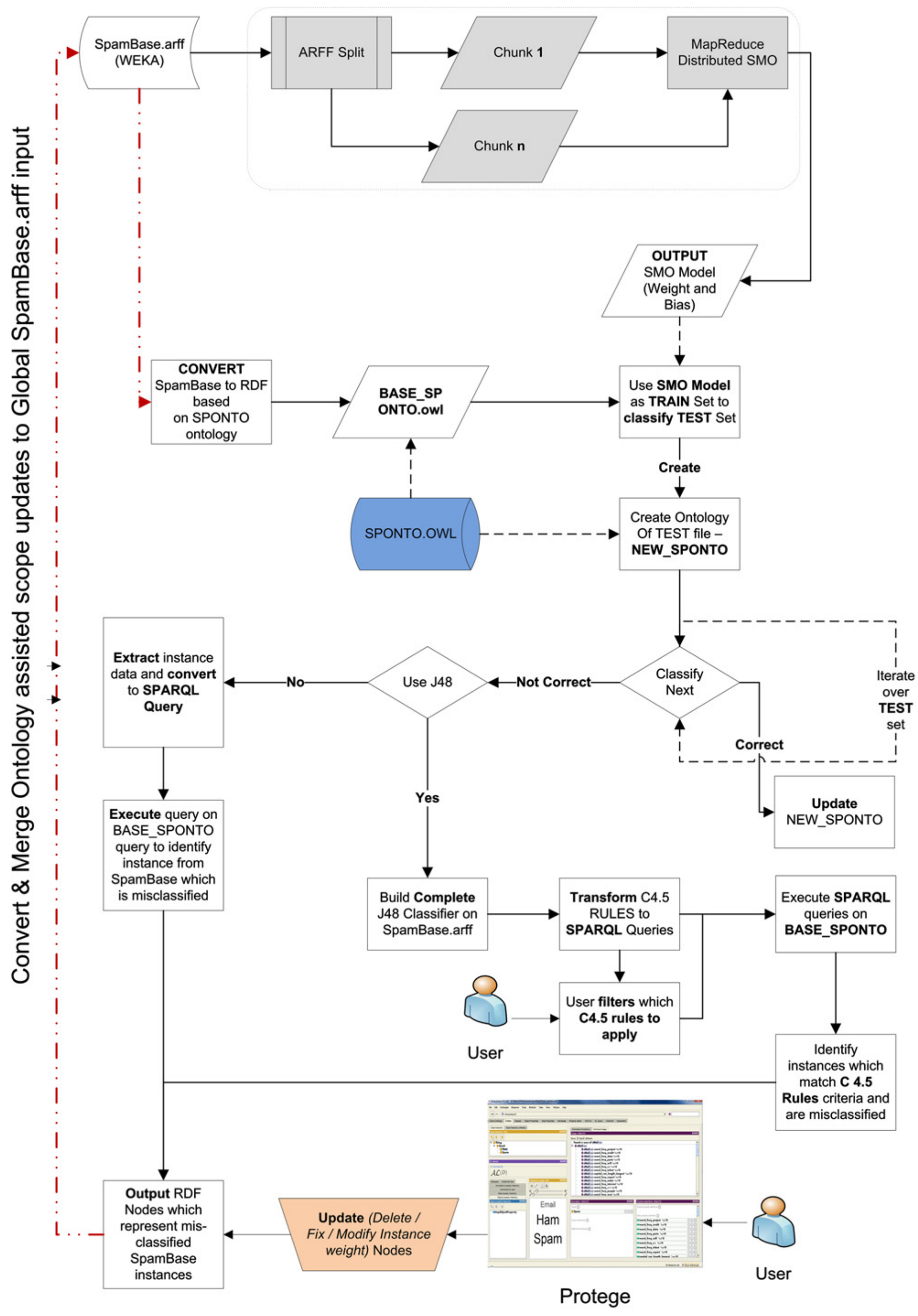

Convert & Merge Ontology assisted scope updates to Global SpamBase.arff input
SpamBase.arff (WEKA)
ARFF Split
Chunk 1
Chunk n
MapReduce Distributed SMO
OUTPUT SMO Model (Weight and Bias)
CONVERT SpamBase to RDF based on SPONTO ontology
BASE_SP ONTO.owl
Use SMO Model as TRAIN Set to classify TEST Set
Create
SPONTO.OWL
Create Ontology Of TEST file – NEW_SPONTO
Extract instance data and convert to SPARQL Query
No
Use J48
Not Correct
Classify Next
Iterate over TEST set
Correct
Update NEW_SPONTO
Yes
Execute query on BASE_SPONTO query to identify instance from SpamBase which is misclassified
Build Complete J48 Classifier on SpamBase.arff
Transform C4.5 RULES to SPARQL Queries
Execute SPARQL queries on BASE_SPONTO
User
User filters which C4.5 rules to apply
Identify instances which match C 4.5 Rules criteria and are misclassified
Output RDF Nodes which represent mis-classified SpamBase instances
Update (Delete / Fix / Modify Instance weight) Nodes
Email
Ham
Spam
Protege
User

Fig. 4. Ontology assisted classification process.

Table 2
Configuration for SMO analysis.

Hardware environment	
Processors	Intel 2.33 GHz Dual Core
Ram	2 GB
Software environment	
SVM	Weka 3.6.0
O/S	Ubuntu 9.10

Polynomial kernel and a subsequent *Gaussian* kernel were carried out. The proposition of the experiments was primarily intended to compare the performance of the parallel approaches. Following the definitions presented in [32], speedup and efficiency in the context of MapReduce can be defined as follows:

$$\text{Speedup} = \frac{\text{Sequential SMO time}}{\text{Parallel SMO time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of processor cores}}$$

The configuration of the MapReduce Hadoop cluster employed for this particular set of experiments is shown in Table 4. Hadoop's default cluster configuration and scheduling is employed. A single TaskTracker is employed on each node. The micro-cluster for this experiment set was made up of 3 physical nodes with a total of 8 CPU cores and 3 virtual nodes each making use of one of the physical cores as summarized below.

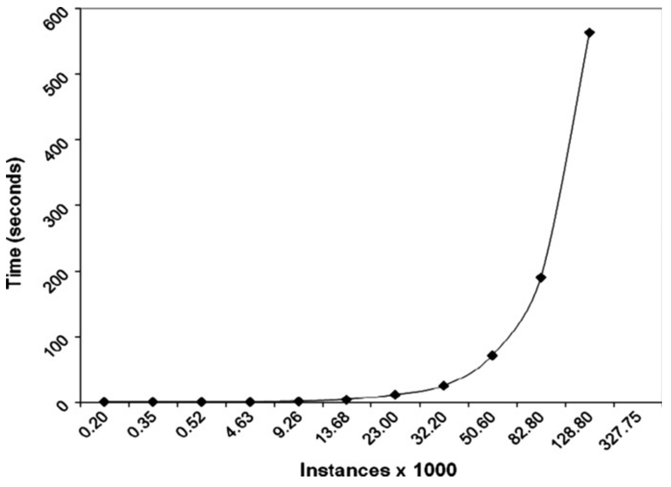


Fig. 5. The training time of the sequential SMO.

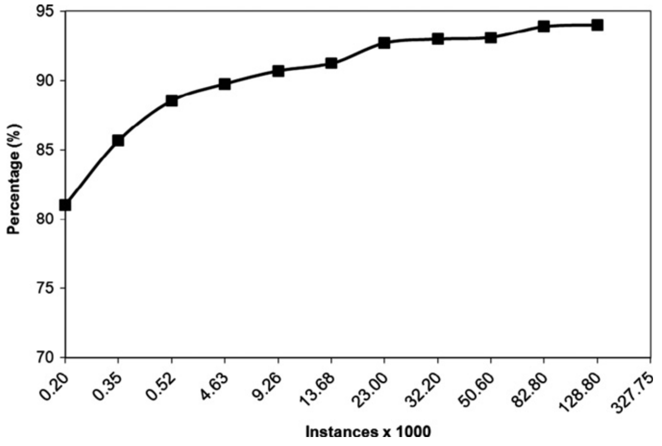


Fig. 6. The accuracy of the sequential SMO.

Table 3
Hadoop cluster configuration.

	CPU	*Ram*
Hardware environment		
Node 1 & 2	Intel Core Duo	2 GB
Node 3	Intel Quad Core	4 GB
Node 4	Virtual Machine on Node 3	512 MB
Software environment		
SVM	Weka 3.6.0 (SMO)	
O/S	Ubuntu 9.10	
Hadoop	Hadoop 0.20	
Java	JDK 1.6	

Fig. 8 and Fig. 9 present the respective *speedup* results of the MapReduce SVM using the 2 kernels.

Fig. 10 and Fig. 11 present the *efficiencies* of the MapReduce SVM using the two kernels, respectively.

For these experiments, we varied the number of MapReduce nodes from 1 to 6 in conjunction with the number of input files, which were adjusted between 4, 8 and 16, for a total of 48,000 instances. Each individual file split is processed by a MapReduce map task. Therefore, for 4 splits, a total of 4 files containing 48,000/4 training instances each using 4 map tasks were processed. For 16 splits, a total of 16 files containing 48,000/16 training instances each using 16 map tasks were processed. The average speedup and efficiency of the MapReduce based SMO algorithm using the Polynomial kernel were about 20 times and 4 times, respectively compared with the sequential SMO. The corresponding results from the MapReduce SMO algorithm using Gaussian kernel were about 28 times and 5 times. From these results we can conclude that the MapReduce based SMO is more efficient than the MPI based approach which has an efficiency of 21/32. The primary reason for this is that the MapReduce based SMO fully distributes the dataset onto a number of computing nodes reducing the overhead in training significantly. In the MPI based

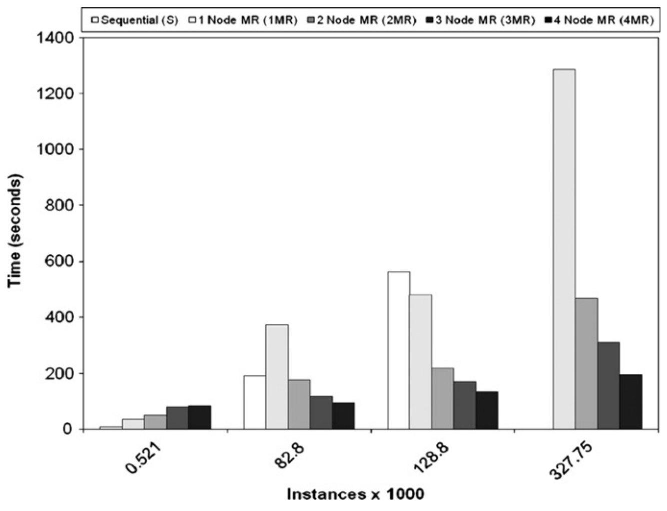


Fig. 7. The efficiency of the parallel SMO.

Table 4
Hadoop configuration.

	CPU	*Cores*	*RAM*
Hardware Environment			
Node 1 & 2	Intel Core Duo	2	2 GB
Node 3	Intel Quad Core	4	4 GB
Node 4, 5 & 6	Virtual Machine on Node 1, 2 &, respectively	1	512 MB
Software Environment			
SVM	WEKA 3.6.0 (SMO)		
OS, Hadoop and Java	Ubuntu 10.04 – Hadoop 0.20.2 – JDK 1.6		

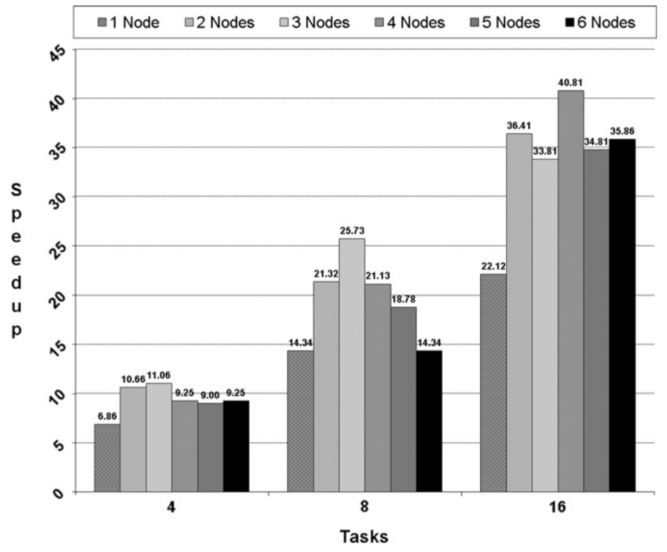


Fig. 8. The speedup of the MapReduce SVM using Polynomial kernel.

approach, only the update computation to the *f*array, b_{up}, $b_{low,}$ i_{up} and i_{low} are performed in parallel. The rest of the SMO algorithm is performed sequentially on a single CPU. Furthermore, the MPI approach also has to deal with the overhead associated with retrieving and converging the global b_{up}, $b_{low,}$ i_{up} and i_{low}, respectively.

5.3. The accuracy of the parallel SMO

Fig. 12 shows that the accuracy of the parallel SMO using 4 MapReduce nodes is comparable to that of the sequential one. Using the global *b* and weight vectors obtained through a MapReduce classifier training run which employed ≈ 4600 instances, the average accuracy of the parallel SMO was $\approx 88\%$ correct in classification. For the case of 327,750 instances, the accuracy was $\approx 92\%$ correct in classification.

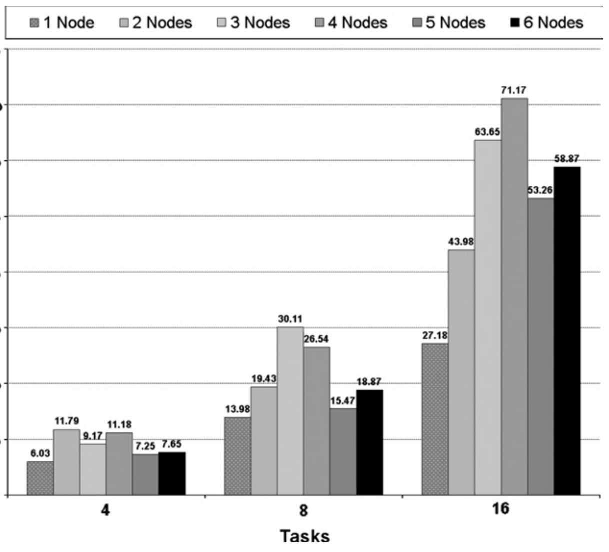


Fig. 9. The speedup of the MapReduce SVM using Gaussian kernel.

Table 5 provides an overall performance comparison between the sequential SMO running on 1 computer and the parallel SMO running in a cluster of 4 MapReduce computers.

Hadoop MapReduce has shown near linear scalability for batch type jobs [62,63]. Given an appropriate number of processing nodes and map tasks, training SVM using the proposed MapReduce approach reduces time considerably. The accuracy of the parallel SMO is comparable to that of the sequential approach. The prototype processes and optimizes each data fragment in parallel using respective *map* operations. The output of each *map* operation reflects the partial weight vector for the localized data fragment. The single reducer sums up the respective partial weight vectors to compute the final global equivalent.

5.4. Evaluating the parallel SMO with ensemble approaches

In machine learning, ensemble schemes provide an opportunity for improving prediction accuracy [64,65]. Bagging and boosting approaches for example are statistically known to improve accuracy in general. However, the degree of actual accuracy improvement (or degradation) depends on the context [66]. Context influencers include classification algorithms, parameterization as well as dataset properties. To explore initial possibilities in this regard, we applied an extension to the prototype and performed a simple experiment to identify any immediate improvements using the proposed MapReduce approach and the SpamBase dataset. Any overall accuracy improvement would have to be considered in the context of any increased computational complexity introduced by the respective processes. We performed two sets of tests. One was based on Weka's bagging (sampling with replacement) method and the other on SMOTE (Synthetic Minority Over Sampling Technique). Bagging [67] involves the random generation of training sets and combining subsequent classifications using the same base classifier.

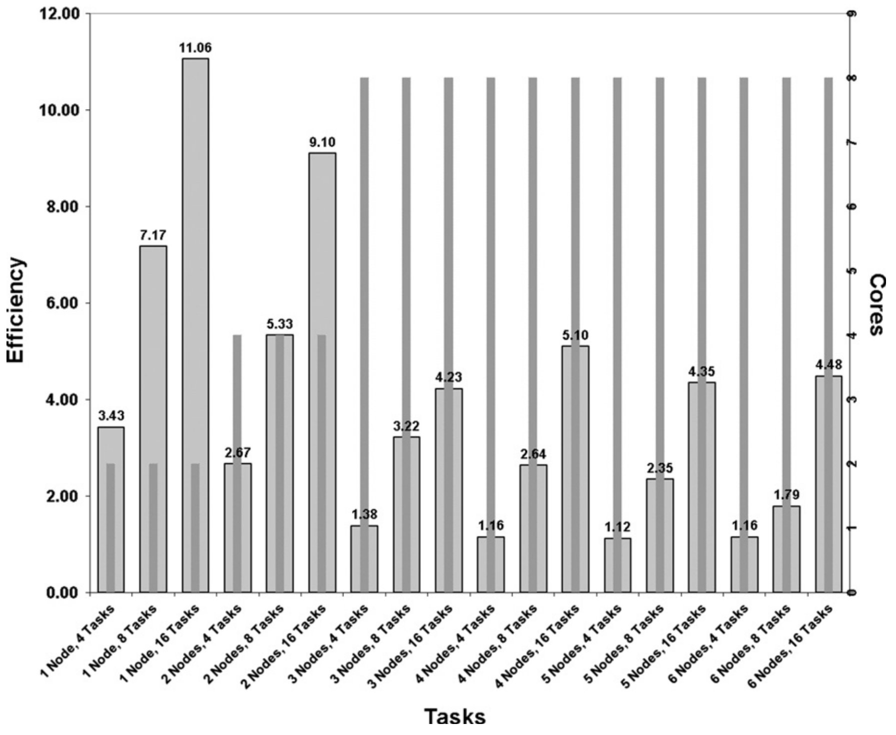


Fig. 10. The efficiency of the MapReduce SVM using Polynomial kernel.

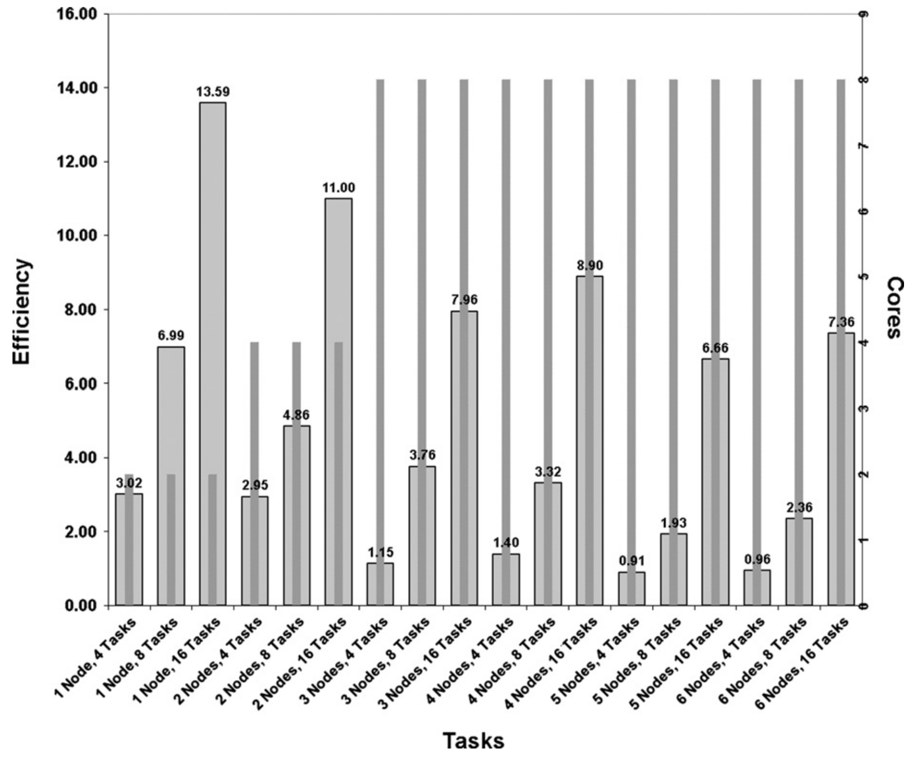


Fig. 11. The efficiency of the MapReduce SVM using Gaussian kernel.

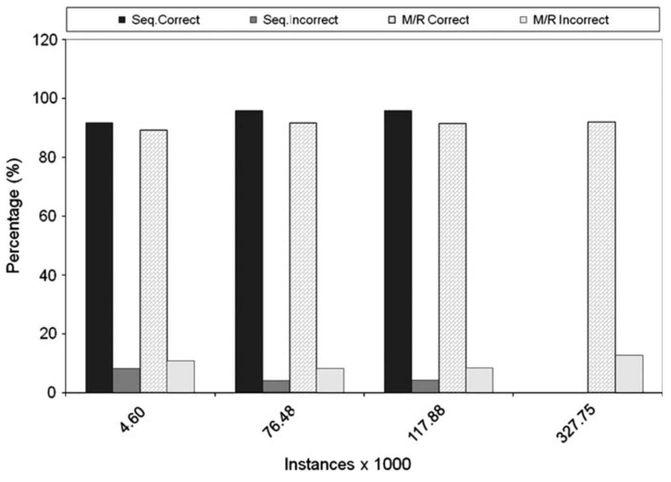


Fig. 12. The accuracy of the parallel SMO.

Table 5
The performance of the parallel SVM using 4 MapReduce nodes.

	Sequential	**MapReduce average** (based on **8** *mappers*)
Correctly classified	≈ 94.03%	≈ 92.04%
Incorrectly classified	≈ 5.97%	≈ 7.96%
128,000 instances (training time in seconds)	≈ 563 s	≈ 134 s

SMOTE [68] focuses on the over-sampling of both minority and majority class. For these tests, 8 input splits (files) were employed using the entire 4600 instances of the SpamBase data set.

Fig. 13 portrays the outcome of the basic evaluation of the application of SMOTE and Bagging. In this exploratory experiment, the MapReduce SMO classifier was seeded with a 100%, 120% and 160% training set over-sample spread across the 8 input splits to evaluate the influence on accuracy using both techniques. The results indicate that in this particular context the recorded accuracy diminished slightly. Using bootstrap aggregation, the parallel SMO classifier achieved a maximum accuracy of 85.7%, reduced to 52.45% when the sampling with replacement target was set to 160%. The corresponding figures for the SMOTE based approach where 85.96% and 80.87%, respectively. Accuracy degradation could be partly influenced by the number of instances evaluated, data and ensemble sizes. These factors have the potential to negatively impact SVM learning when using ad-hoc ensemble techniques. Accuracy in this context is also influenced by the current bias and weight aggregation approach in the respective Reducer phase of MapReduce.

5.5. *Ontology augmentation*

As described in Section 4, to further improve the accuracy of the MapReduce based parallel SMO, we augment the base training sets with additional intelligence through ontology based semantics. This is performed by influencing individual misclassified elements as well as attributing increased weighting to user identified and selected instances. Fig. 14 shows the percentage of accuracy improvement across the number of file splits by employing the ontology based approach. There is an average of ≈ 5% accuracy improvement, ranging from a minimum of 1.7% when the file splits are 4 to a maximum of 7.5% when the number of input files is 48.

Fig. 15 presents a comparative analysis of the rate of accuracy degradation between the ontology augmented approach (X-SpamBase) and the original (SpamBase)—the former rate is significantly slower when increasing the number of training input file splits.

Based on an average accuracy improvement of 4.6% over the baseline parallel SMO, Fig. 16 shows that using the ontology augmented approach, the MapReduce based parallel SMO achieves a maximum accuracy of ≈ **99**% and an average of 96%,

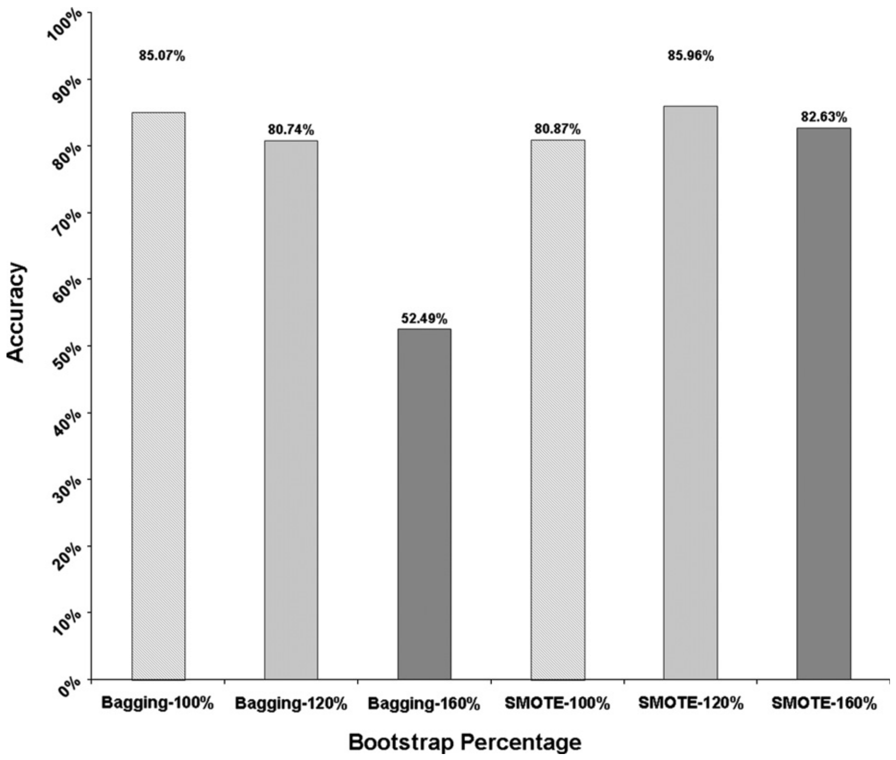


Fig. 13. The effect of Bagging and SMOTE.

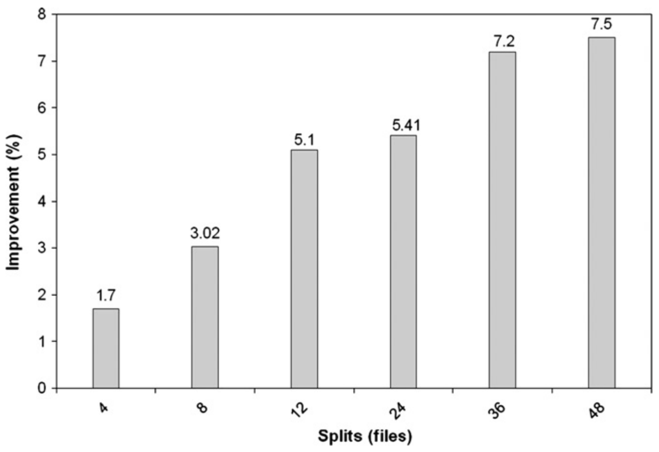


Fig. 14. The impact of file splits on accuracy improvement.

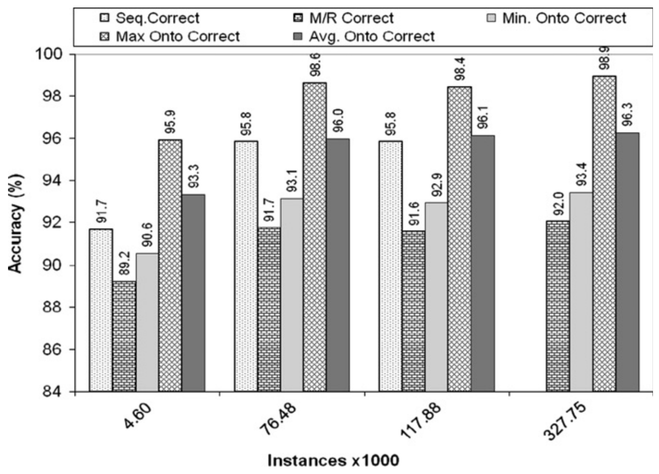


Fig. 16. The accuracy of the ontology augmented MapReduce SMO.

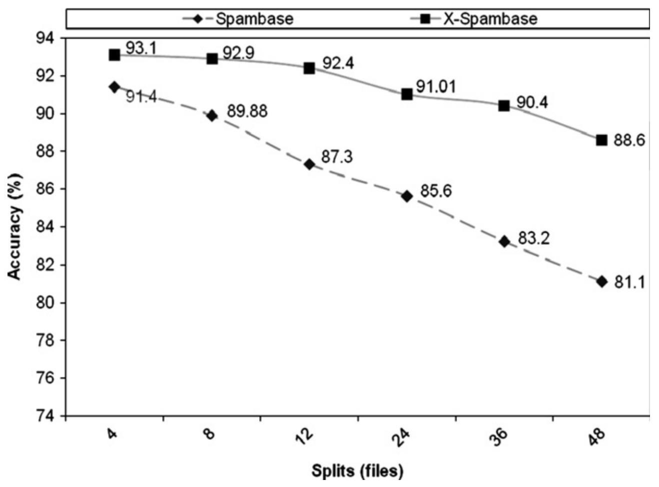


Fig. 15. The impact of ontology augmentation on accuracy degradation.

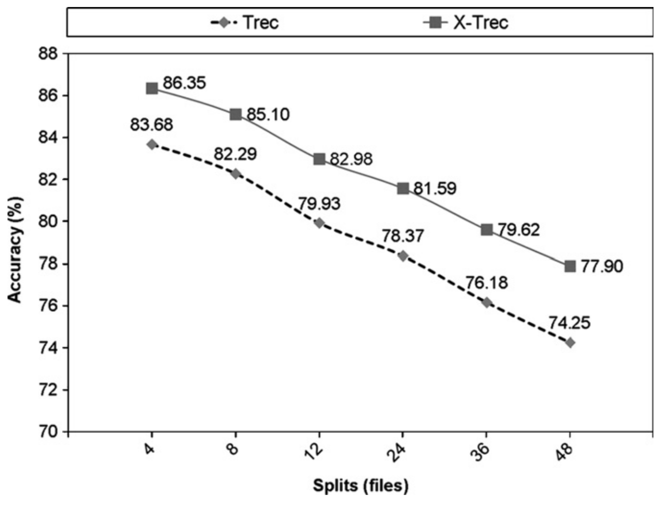


Fig. 17. The impact of ontology augmentation on classification accuracy on the TREC data set.

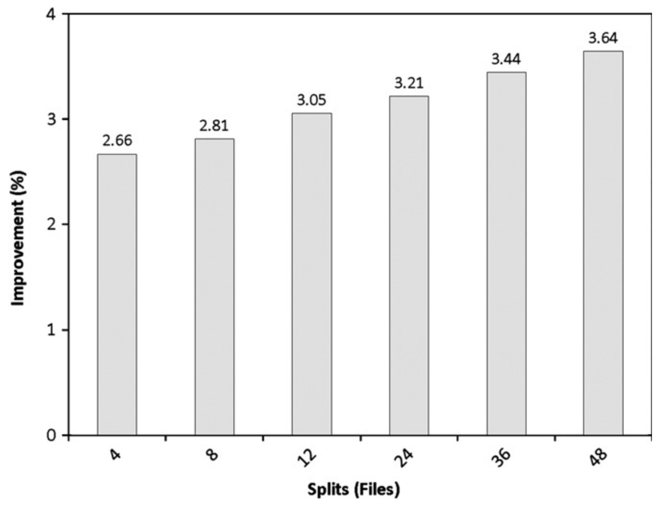


Fig. 18. The impact of TREC data splits on classification accuracy improvement.

which performs better than the original sequential SMO in classification.

We have also converted a large TREC data set [69] to a format similar to SpamBase for further evaluation of the ontology augmented approach. The TREC corpus is constituted of about 75,000 messages out of which 50,000 messages are spam. An initial experimental assessment shows similar accuracy improvement over the MapReduce based parallel SMO as portrayed in Fig. 17 and Fig. 18, respectively, where '*Trec*' represents the classification accuracy of the parallel SMO on the TREC data set and '*X-Trec*' represents the accuracy of the corresponding ontology augmented approach.

6. Conclusions and future work

In this paper we have presented a MapReduce based parallel SVM algorithm for fast spam filter training. By distributing the data set into a number of computing nodes, the parallel SVM reduces the training time considerably. To mitigate accuracy degradation in classification, the parallel SVM is augmented with ontology semantics.

There is ample room for further improvement to the parallel SVM. We intend to research appropriate schemes on how to automatically extract additional intelligence from annotated instances and employ this with the feedback loop to the machine learning process within the parallel SVM. Currently, the ontology based feedback loop approach is mostly based and capitalizes on human expertise to identify hidden context which mitigates the problem of concept drift. We intend to research relevant techniques to automatically identify concept drift in classification similar to the work presented in [42].

References

[1] E. Blanzieri, A. Bryl, A survey of learning-based techniques of email spam filtering, Artif. Intell. Rev. 29 (1) (2008) 63–92.
[2] Á. Blanco, A. Ricket, M. Martín-Merino, F. Sandoval, A. Prieto, J. Cabestany, M. Graña, Computational and Ambient Intelligence, 4507, Berlin, Heidelberg: Springer, Berlin Heidelberg, 2007 903–910.
[3] K.-L. K. Li, H.-K. Huang, S.-F. Tian, L. Kun-Lun, L. Kai, H. Hou-Kuan, and T. Sheng-Feng, Active learning with simplified SVMs for spam categorization, in: Proceedings. International Conference on Machine Learning and Cybernetics, vol. 3, pp. 1198–1202, 2002.
[4] L. Mei, Y. Cheng-qing, Anti-spam technique with SVM and K-NN cooperation method, Comput. Eng. Appl. 44 (4) (2008) 135–137.
[5] L. Zhang, J. Zhu, T. Yao, An evaluation of statistical spam filtering techniques, ACM Trans. Asian Lang. Inf. Process. (TALIP) 3 (4) (2004) 243–269.
[6] H.P. Graf, E. Cosatto, L.L. Bottou, I. Durdanovic, V. Vapnik, Parallel support vector machines: the cascade svm, In Adv. Neural Inf. Process. Syst. (2005) 521–528.
[7] G. Huang, K.Z. Mao, C. Siew, D. Huang, S. Member, Fast modular network implementation for support vector machines, IEEE Trans. Neural Networks 16 (2005) 1651–1663.
[8] T.-N. Do and F. Poulet, Classifying one billion data with a new distributed svm algorithm, in: Proceedings of the International Conference on Research, Innovation and Vision for the Future, pp. 59–66, 2006.
[9] E.Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, and Z. Qiu, PSVM: parallelizing support vector machines on distributed computers, in: Proceedings of the 21st Annual Conference on Neural Information Processing Systems, 2007.
[10] N. Cristianini, J. Shaw-Taylor, An Introduction to SVM's and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
[11] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schölkopf, C.J.C. Burges, A.J. Smola (Eds.), Advances in Kernel Methods: Support Vector Machines, 1999, pp. 185–208.
[12] F.R. Bach and M.I. Jordan, Predictive low-rank decomposition for kernel methods, in: Proceedings of the 22nd International Conference, pp. 33–40, 2005.
[13] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Process. Lett. 9 (3) (1999) 293–300.
[14] B. Catanzaro, N. Sundaram, and K. Keutzer, Fast support vector machine training and classification on graphics processors, in: Proceedings of the 25th International Conference on Machine Learning—ICML'08, pp. 104–111, 2008.
[15] T.-N. Do, V.-H. Nguyen, F. Poulet, Advanced Data Mining and Applications, Berlin Heidelberg, vol. 5139, Berlin, Heidelberg: Springer, 2008, pp. 147–157.
[16] H. Yu, J. Yang, and J. Han, Classifying large data sets using SVMs with hierarchical clusters, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD'03, pp. 306–315, 2003.
[17] J. Dean, S. Ghemawat, MapReduce, Commun. ACM 51 (1) (2008) 107, Jan.
[18] N. A. Syed, S. Huan, L. Kah, and K. Sung, Incremental Learning with Support Vector Machines, 1999. [Online]. Available: 〈http://citeseer.ist.psu.edu/view doc/summary?doi=10.1.1.46.6367〉. (accessed: 03-Aug-2010).
[19] G. Caruana, M. Li, and H. Qi, SpamCloud: A MapReduce based anti-spam architecture, in Proceedings of the Seventh International Conference on Fuzzy Systems and Knowledge Discovery, vol. 6, pp. 3003–3006, 2010.
[20] G. Caruana, M. Li, and M. Qi, A MapReduce based parallel SVM for large scale spam filtering, in: Proceedings of the Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2659–2662, 2011.
[21] E. Prud'hommeaux and A. Seaborne, SPARQL Query Language for RDF (Working Draft), 〈http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123〉, 2005. (accessed:07-Oct-2012).
[22] S.C. for B. I. Research, Protege [Online]. Available: 〈http://protege.stanford.edu/〉 (accessed: 01-Oct-2011).
[23] G. Caruana, M. Li, A survey of emerging approaches to spam filtering, ACM Comput. Surv. 44 (2) (2012) 1–27.
[24] PoweredBy—Hadoop Wiki. [Online]. Available: 〈http://wiki.apache.org/hadoop/PoweredBy〉.
[25] RSA Conference | Connect | HT1-301: Yokai vs. the Elephant: Hadoop and the Fight Against Shape-Shifting Spam (PK Session). [Online]. Available: 〈http://365.rsaconference.com/community/connect/blog/2010/02/12/ht1-301-yokai-vs-the-elephant-hadoop-and-the-fight-against-shape-shifting-spam-pk-session〉. (accessed: 28-Jul-2012).
[26] A.S. Das, M. Datar, A. Garg, and S. Rajaram, Google news personalization, in: Proceedings of the 16th International Conference on World Wide Web—WWW '07, pp. 271–280, 2007.
[27] M.G. Noll and C. Meinel, Building a scalable collaborative web filter with free and open source software, in: Proceedings of the IEEE International Conference on Signal Image Technology and Internet Based Systems, pp. 563–571, 2008.
[28] F. Colas, P. Brazdil, M. Bramer, Artificial Intelligence in Theory and Practice, 217, Springer, US, 2006 169–178.
[29] M. Ye, T. Tao, F.-J. Mai, and X.-H. Cheng, A spam discrimination based on mail header feature and SVM, in: 2008 Fourth International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4, 2008.
[30] B. Scholkopf and A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, 2001.
[31] G. Zanghirati, L. Zanni, A parallel solver for large quadratic programs in training support vector machines, Parallel Comput. 29 (4) (2003) 535–551.
[32] L.J. Cao, S.S. Keerthi, C.-J.J. Ong, J.Q. Zhang, U. Periyathamby, X.J.J. Fu, H.P. Lee, Parallel sequential minimal optimization for the training of support vector machines, IEEE Trans. Neural Networks 17 (4) (2006) 1039–1049.
[33] K. Woodsend, J. Gondzio, Hybrid MPI/OpenMP parallel linear support vector machine training, J. Mach. Learn. Res. 10 (2009) 1937–1953.
[34] B. C. Kuszmaul, Cilk provides the 'best overall productivity' for high performance computing, in: Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures—SPAA'07, pp. 299–300, 2007.
[35] J. Yang, X. Yang, and J. Zhang, A parallel multi-class classification support vector machine based on sequential minimal optimization, in: Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06), pp. 443–446, 2006.
[36] C.-T. T. Chu, S. K. Kim, Y.-A. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, Map-reduce for machine learning on multicore, in: Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS), pp. 281–288, 2006.

[37] J. Kim, D. Dou, H. Liu, D. Kwak, Advances in Artificial Intelligence, 4509, Springer, Berlin, Heidelberg, 2007, pp. 272–283.
[38] I.H. Witten, E. Frank, and M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 2011.
[39] S. Youn and D. McLeod, Efficient spam email filtering using adaptive ontology, in: Proceedings of the Fourth International Conference on Information Technology, pp. 249–254, 2007.
[40] M. Balakumar and V. Vaidehi, Ontology based classification and categorization of email, in: Proceedings of the International Conference on Signal Processing, Communications and Networking, pp. 199–202, 2008.
[41] A. Tsymbal, The Problem of Concept Drift: Definitions and Related Work, Citeseer, Ireland, Apr. 2004.
[42] A. Han, H.-J. Kim, I. Ha, and G.-S. Jo, Semantic analysis of user behaviors for detecting spam mail, in: Proceedings of the IEEE International Workshop on Semantic Computing and Applications, pp. 91–95, 2008.
[43] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[44] W.N. Gansterer, A.G.K. Janecek, R. Neumayer, M.W. Berry, M. Castellanos, Survey of Text Mining II, Springer, London, 2008, pp. 165–183.
[45] G. Gargiulo, A. Picariello, and C. Sansone, Using Visual and Semantic features for anti-spam filtering, in: Proceedings of 21st Neural Information Processing Systems, 2007.
[46] J.-H. Hsia and M.-S. Chen, Language-model-based detection cascade for efficient classification of image-based spam e-mail, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), pp. 1182–1185, 2009.
[47] S. Youn and D. McLeod, Improved spam filtering by extraction of information from text embedded image e-mail, in: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), pp. 1754–1755, 2009.
[48] B. He, W. Fang, Q. Luo, N.K. Govindaraju, and T. Wang, Mars: a MapReduce framework on graphics processors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 260–269, 2008.
[49] K. Taura, K. Kaneda, T. Endo, A. Yonezawa, Phoenix, a parallel programming model for accommodating dynamically joining/leaving resources, ACM SIGPLAN Not. 38 (10) (2003) 216–229.
[50] Welcome to Apache Hadoop! [Online]. Available: 〈http://hadoop.apache.org/〉. (accessed: 07-Oct-2012).
[51] T. Aarnio, Parallel Data Processing with Mapreduce, TKK T-110.5190, Seminar on Internetworking, 2009. [Online]. Available: 〈http://www.cse.tkk.fi/en/publications/B/5/papers/Aarnio_final.pdf〉. (accessed: 07-Oct-2012).
[52] A. Anand, Apache Hadoop Wins Terabyte Sort Benchmark. [Online]. Available: 〈http://developer.yahoo.net/blogs/hadoop/2008/07/apache_hadoop_wins_terabyte_sort_benchmark.html〉. (accessed: 07-Oct-2012).
[53] O. O'Malley, Terabyte Sort on Apache Hadoop, Available: 〈http://sortbenchmark.org/YahooHadoop.pdf〉. (accessed: 07-Oct-2012).
[54] V. Vapnik, A. Lerner, Pattern recognition using generalized portrait method,, Autom. Remote Control 24 (1963).
[55] M. Kearns, Efficient noise-tolerant learning from statistical queries, J. ACM 45 (6) (1998) 983–1006.
[56] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software,, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 10–18.
[57] B.E. Boser, I.M. Guyon, and V.N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT), pp. 144–152, 1992.
[58] A. Asuncion and D.J. Newman, UCI Machine Learning Repository, available online: http://archive.ics.uci.edu/ml/ (accessed: 07-Oct-2012).
[59] Matthew Horridge and P.F. Patel-Schneider, Manchester Syntax OWL 2 Web Ontology Language, W3C, 2011. [Online]. Available: 〈http://www.w3.org/2007/OWL/wiki/ManchesterSyntax〉. (accessed: 07-Oct-2012).
[60] U. Rebbapragada and C.E. Brodley, Class noise mitigation through instance weighting, in: Proceedings of the 18th European Conference on Machine Learning (ECML), vol. 4701, pp. 708–715, 2007.
[61] A. Frank and A. Asuncion, UCI Machine Learning Repository—Adult Data Set, [Online] 〈http://archive.ics.uci.edu/ml/datasets/Adult〉 (accessed: 07-Oct-2012).
[62] A. Pavlo, E. Paulson, A. Rasin, D.J. Abadi, D.J. DeWitt, S. Madden, and M. Stonebraker, A comparison of approaches to large-scale data analysis, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 165–178, 2009.
[63] 5 Common Questions About Hadoop «Cloudera» Apache Hadoop for the Enterprise. [Online]. Available: 〈http://www.cloudera.com/blog/2009/05/5-common-questions-about-hadoop/〉. (accessed: 07-Oct-2012).
[64] E. Bauer, R. Kohavi, An empirical comparison of voting classification algorithms: bagging, boosting, and variants,, Mach. Learn. 36 (1–2) (1999) 105–139.
[65] W.N. Street and Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377–382, 2001.
[66] S. Wang, A. Mathew, Y. Chen, L. Xi, L. Ma, J. Lee, Empirical analysis of support vector machine ensemble classifiers,, Expert Syst. Appl. 36 (3) (2009) 6466–6476.
[67] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[68] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (1) (2002) 321–357.
[69] E. M. Voorhees and L. P. Buckland, Overview of TREC 2007, in: Proceedings of the 16th Text REtrieval Conference (TREC), 2007.

Godwin Caruana holds a Master of Science in Information Technology and is currently a PhD research student in the School of Engineering and Design at Brunel University, UK. His research interests are in the areas of distributed computing, data mining and applications such as spam filtering, intelligent systems. He has over 20 years experience in IT research, development and deployment. He is a member of ACM, IEEE, the BCS and a Chartered IT Professional.

Maozhen Li is a Senior Lecturer in the School of Engineering and Design at Brunel University, UK. He received the PhD from Institute of Software, Chinese Academy of Sciences in 1997. His research interests are in the areas of intelligent system, semantic web, parallel and distributed computing technologies and applications (grid computing, cloud computing, peer-to-peer computing). He has about 100 scientific publications in these areas. He has served over 30 international conferences. He is currently on editorial boards of three international journals including Journal of Cloud Computing Advances, Systems and Applications, Springer. He is a fellow of the British Computer Society.

Yang Liu received the PhD in the School of Engineering and Design at Brunel University, UK in May 2011. His research interests include distributed computing, grid computing, data mining, information retrieval.