Information Sciences 232 (2013) 419–436

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

An efficient classification approach for large-scale mobile ubiquitous computing

Feilong Tang [a,*], Ilsun You [b], Can Tang [c], Minyi Guo [d]

[a] School of Software, Shanghai Jiao Tong University, Shanghai 200240, China
[b] School of Information Science, Korean Bible University, Seoul, South Korea
[c] Department of Finance, Heilongjiang University, Harbin 150080, China
[d] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

ARTICLE INFO

Article history:
Received 31 August 2011
Received in revised form 30 August 2012
Accepted 20 September 2012
Available online 16 October 2012

Keywords:
Classification
Semi-sparse algorithm
Multi-class multi-label classification
Parallelization
Sequential minimal optimization (SMO)
Mobile ubiquitous computing

ABSTRACT

Context classification is at the center of user-centric ubiquitous computing that targets the provision of personalized services based on expressed preferences and interests. Classification of context for Mobile Ubiquitous Computing (MUC), where there are high volumes of data and users place large demands on a context-aware system, must be effective and efficient in computational terms. The Sequential Minimal Optimization (SMO) based SVMTorch is widely used for text classification; it is however inefficient for MUC-oriented context analysis due to: (1) a low classification speed caused by inefficient matrix multiplication, and (2) the inability to classify multi-label data.

In this paper, we propose an efficient classification approach to improve and extend the SVMTorch. Firstly, we propose a semi-sparse algorithm to speed up vector/matrix multiplication which lies at the core of the SVMTorch-based classification approaches. Theoretically, to multiply two vectors (i.e., a selected vector and a trained vector) with m and n non-zero elements respectively, the traditional SVMTorch needs $O(m + n)$ time while our semi-sparse algorithm requires only $O(n)$ time, where n is the number of non-zero elements in the trained vector. Secondly, we extend the functions of the traditional SVMTorch approach which is limited to the classification of single-label data, to support multi-class multi-label classification. Finally, we parallelize the improved SVMTorch which incorporates the semi-spares algorithm and function extensions to access multi-core processor and cluster systems to further improve the effectiveness and efficiency of the classification process. The experimental results demonstrate that our proposed solution significantly improves the performance and capability of the traditional SVMTorch. The results support the conclusion that the larger training and testing data sets are, the more improvement our solution brings to the effectiveness and efficiency of the context classification. This conclusion is verified in a Chinese web page classifier developed based on the solution presented in this paper.

1. Introduction

Ubiquitous computing is a user-centric mobile computing paradigm in which users enjoy service provision in a dynamic mobile situation [24,46]. To provide mobile users with preferred services adaptively classification based on context is a prerequisite in many ubiquitous applications; for example, in recommender systems the goal is targeted service provision of resources to specific users based on h/her context. However, in *Mobile Ubiquitous Computing* (MUC) environments there

* Corresponding author. Tel.: +86 2134202949; fax: +86 2134204728.
E-mail address: tang-fl@cs.sjtu.edu.cn (F. Tang).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.09.050

are serious challenges to the classification of text due to: (1) *large-scale and high-dimensional contextual data*, and (2) *real-time implementation where users have highly dynamic spatial change*. Consequently, context classification approaches for MUC applications should be sufficiently fast and versatile to meet the highly dynamic environments and provide support for multi-class multi-label context analysis [11,33,44].

Among existing classification models, the machine learning based support vector machine (SVM) outperforms other methods especially in high-dimensional nonlinear classification [2,34,39]. SVMTorch is a classic implementation of the SVM. It exhibits excellent performance on text classification, and has been widely used in face recognition, image processing and other areas. However, for MUC environments with the characteristics identified in points 1 and 2 above the SVMTorch fails to perform effectively due to a low classification speed caused by inefficient matrix multiplication and the inability to implement multi-label classification.

The *Sequential Minimal Optimization* (SMO) algorithm forms the kernel of the SVMTorch. Different from other classification methods that solve the largest possible optimization problem at each step, the SMO solves the smallest possible optimization problem in each iteration. Specifically, the SMO chooses two Lagrange multipliers to jointly optimize for these multipliers and updates the SVM to reflect new optimal values at each step [9,25]. In traditional implementations (e.g., SVMTorch) of the SMO, there is a large number of sparse matrix multiplication for not only the linear kernel but also the Gaussian kernel. We observe that during the matrix multiplication, the selected sparse vector is thoroughly traversed in each iteration, which inevitably results in time-consuming training and testing processes, especially to large-scale sparse matrices. On the other hand, SVMTorch only supports single-label classification, therefore it is incapable of classifying MUC-oriented contextual data with multiple labels.

The motivation of this paper is set to investigate and solve the identified challenging issues. The solution proposed in this paper improves matrix multiplication efficiency of the SVMTorch and extends its functions to support multi-class multi-label data classification in multi-core processor and cluster systems. The main contributions of this paper are summarized as follows.

- We propose a semi-sparse algorithm for vector/matrix multiplication to improve the traditional SMO algorithm. Traversing a selected vector at each iteration in the SMO wastes a large amount of training and testing time during the sparse matrix multiplication. Our semi-sparse algorithm can speed up the training and testing processes of the SVMTorch classifier through *densifying* the selected sparse vector to avoid redundant traverses. A theoretical analysis indicates that the traditional SMO requires $O(m + n)$ time on traversing two vectors (a selected vector and a trained vector) with m and n non-zero elements in a vector multiplication. Our semi-sparse algorithm improves efficiency in that it only needs $O(n)$ time for vector multiplication without reducing classification accuracy of the SVMTorch. The experimental results also show that the two implementations of our semi-sparse algorithm can reduce the training time to 53.7% and 74.95% of that in the traditional SVMTorch respectively.
- We extend the traditional SVMTorch classifier to support multi-label data classification. The traditional SVMTorch has a good behavioral characteristics in two-class and multi-class single-label classifications; however, it cannot support multi-class multi-label classification tasks. This solution proposed in this paper transforms a multi-label problem into multiple single-label sub-problems based on the 1-vs-many policy and proposes a comprehensive evaluation metrics. Experimental results demonstrate that our approach exhibits excellent precision and performance for multi-class multi-label classification tasks.
- We parallelize the SVMTorch classifier. This demonstrates enhanced performance in both the semi-sparse matrix multiplication and multi-label classification in multi-core processor and cluster systems. Existing SVMTorch implementations are restricted to single-core processor systems; this limits its classification speed. We use MPI (Message Passing Interface) to simultaneously distribute multiple sub-tasks in a training or testing process to different cores; this enables a significant increase in the speed of the classification process significantly.
- A Chinese web page classifier has been developed predicated on our novel semi-sparse algorithm, parallelization technology and multi-label classification approach. It has been shown to work very well in large-scale Chinese web page classification.

The remainder of this paper is organized as follows. In Section 2, we introduce preliminaries with a discussion on the background. In Section 3, we present our novel semi-sparse algorithm for vector/matrix multiplication with an analysis. Section 4 presents our proposed approach to multi-class multi-label context classification and the parallelization scheme. Experimental results and comprehensive performance evaluations are reported in Section 5. The paper closes with conclusions set out in Section 6.

2. Preliminaries and background

In this section we initially introduce some preliminaries followed by a review related research.

2.1. Preliminaries

The SVM, developed by Vapnik [43], exhibits many attractive features including high classification precision. The SVM solves classification problems generally through a quadratic optimization. Let there be a training set with ℓ samples

$(x_i, y_i)(x_i \in E, y_i \in R)$ and, where E is an Euclidean space with a scalar dot product. The optimization problem in the SVM can be formulated as follows [30].

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} y_i y_j k(\vec{x_i}, \vec{x_j}) \alpha_i \alpha_j$$

$$0 \leqslant \alpha_i \leqslant C, \quad \forall i$$

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0$$

A point is optimal if and only if Karush–Kuhn–Tucker (KKT) conditions are fulfilled and $Q_{ij} = y_i y_j k(\vec{x_i}, \vec{x_j})$ is positive semi-definite [41]. Such the point may be non-unique and non-isolated. The KKT conditions are particularly KKT simple. The quadratic programming problem can be solved using the following formula (for all i).

$$\alpha_i = 0 \Rightarrow y_i f(\vec{x_i}) \geqslant 1$$

$$0 \leqslant \alpha_i \leqslant C \Rightarrow y_i f(\vec{x_i}) = 1$$

$$\alpha_i = C \Rightarrow y_i f(\vec{x_i}) \leqslant 1$$

In this way, solutions to SVM optimization problems are changed into how to obtain optimal values for the above quadratic programming problem; this can be achieved through a large volume of sparse matrix multiplications during a classification process. In 1998, Platt proposed the famous SMO algorithm for solving the sparse matrix multiplication problem in SVM optimization. The SMO algorithm reduces the size of data sets to 2 samples at each step in the classification process and identifies the solution iteratively. It is well-known that high-dimensional matrix multiplication is very time-consuming, occupying most training and testing time in SVM-based SMO algorithms. Therefore, the speed of high-dimension vector/matrix multiplication is an important factor in the classification time. In actuality, it is the time taken by SMO-based classifiers which restricts their application in large-scale text classification problems.

2.2. Background

Classification is one of the most important data mining tasks [1,13,17,23,28,32] and has been widely researched in recent years [3,7,16,26,42]. As a classic classification method, the SVM has advantages such as: adequate generalization to new objects, absence of local minima, and representation that depends on only a few parameters [8,31]. Most SVM-based training algorithms, for example: Chunking in [4], the decomposition method [35–37], Shrinking [21], BSVM [18] and Alpha Seeding [15], use the decomposition technology. The GBT algorithm proposed by Joachims is an extension the Osuna algorithm. An implementation of this algorithm is termed SVMLight [22].

Sequential minimal optimization (SMO), proposed by John Platt in 1998 [41], is possibly the most important representative among decomposition-based classifiers. This algorithm is the kernel of the SVMTorch which is widely used for data classification and regression. The SMO always solves the smallest possible optimization problem through optimizing two Lagrange multipliers at each step.

In recent years many application-specific solutions have been proposed for solving specified classification tasks [5,10]. In [29] Ma et al. proposed an approach for gene classification adopting codon usage bias as feature inputs. The DNA sequence is first converted to a feature vector with 59 dimensions, where each element corresponds to the Relative Synonymous Usage (RSCU) frequency of a codon. The Support-Vector-Based Fuzzy Neural Network (SVFNN) [27] combines the superior classification power of SVM in high dimensional data spaces and the efficient human-like reasoning of Fuzzy Neural Networks (FNN) to handle uncertainty in pattern classification. F^2SVM (fuzzy-input fuzzy-output SVM) [6] was designed for subpixel image classification. This binary classifier can address multi-class problems using two strategies: the fuzzy one-against-all (FOAA) and the fuzzy one-against-one (FOAO) strategies. It can process inputs of the classification algorithm for modeling the subpixel information in the learning phase and provide a fuzzy model of classification results, allowing a many-to-one relationship among classes and pixels. Personalized Transductive Learning (PTL) [38] builds a unique local model for classification of individual test samples and is practically Dependant on neighborhood, using two concepts: knowledgeable neighborhood and transductive SVM classification tree (t-SVMT). The Support Feature Machine (SFM) [12] is a multidimensional time series classification model. It uses the optimization model of SVM and the nearest neighbor rule to incorporate both spatial and temporal features of multi-dimensional data. In [19], the authors proposed an enhanced hybrid classification method through the utilization of the naive Bayes approach and the SVM. The Bayes formula is used to vectorize a document according to a probability distribution reflecting the probable categories to which documents may belong. The SVM can then be used to classify the documents on a multidimensional level.

The Twin SVM (TWSVM)[20] is a binary SVM classifier which determines two nonparallel planes by solving two related SVM-type problems, each of which is smaller than in a conventional SVM. The TWSVM solves two quadratic programming problems of a smaller size rather than a large one in traditional SVMs. The Twin Mahalanobis Distance-Based Support Vector

Machine (TMSVM) [40] extends the TWSVM by constructing two Mahalanobis distance-based kernels according to the covariance matrices of two classes of data for optimizing the nonparallel hyperplanes. As a result, it is suitable for data with different covariance matrices.

A complete framework for XML document classification to capture structures and contents of XML-based data has been proposed in [45]. In this work, the author designs a knowledge representation method for XML documents with a decision-tree learning algorithm for the XML classification problem and a semi-supervised learning algorithm. An Adaptive Classification System (ACS) is presented in [14]. ACS has been developed for video-based face recognition and it combines a fuzzy ARTMAP neural network classifier with a dynamic particle swarm optimization (DPSO) algorithm, and a has long term memory.

3. Semi-sparse matrix multiplication

The SMO algorithm is the core of the SVMTorch. Vector/matrix multiplication is a basic operation for iteration-based data classification in the SMO. Consequently, vector/matrix multiplication speed significantly impacts the training time of the SVMTorch. The SMO algorithm in traditional SVMTorch generally uses sparse vector multiplication for data training. During each iteration, both the selected vector and trained vector are repeatedly traversed, which wastes a lot of training time. In this section, we propose the semi-sparse algorithm to expedite the matrix multiplication process, and then analyze the performance of our algorithm.

3.1. Motivation

Algorithm 1. Sparse matrix multiplication in the traditional SMO

```
1: i = 0, p1 = 0, p2 = 0, dot = 0;
2: while (i < N_V) {
3:    while (p1 < num1 && p2 < num2) {
4:       a1 = index[p1];
         a2 = index[p2];
5:       if (a1 == a2) {
6:          dot += value[p1] * value[p2];
            p1++;
            p2++;}
7:       else if (a1 > a2)
                p2++;
8:            else
                p1++;
      }
9:   result[i]=dot;
10:  i++;}
```

In traditional SMO algorithm, trained samples are stored in sparse matrix, where each sparse vector consists of only non-zero elements and is stored in two arrays: *index[]* and *value[]*. The *index[]* is an integer array that stores locations of non-zero elements, and the *value[]* is a float point array that stores the corresponding non-zero value.

The concept of matrix multiplication in the traditional SMO algorithm is shown in Algorithm 1, where index[p1] and index[p2] point to non-zero elements in a selected vector p1 and a trained vector p2, respectively; similarly, value[p1] and value[p2] store the non-zero values corresponding to index[p1] and index[p2], respectively. N_V is the number of trained vectors in a matrix. *num1* and *num2* are the numbers of the non-zero elements in p1 and p2, respectively.

To multiply two sparse vectors p1 and p2, Algorithm 1 needs to find the same subscript through traversing the two vectors. Assume there be a set of vectors in a training matrix. To get the optimal value, all of them need to multiply by the selected vector. Consequently, the selected vector in Algorithm 1 will be traversed N_V times in the above matrix multiplication process. When the trained vectors are high-dimensional, the traditional SMO algorithm will take insufferable time for a training or a testing.

Based on this analysis, we propose a new semi-sparse algorithm to speed up the matrix multiplication.

3.2. Semi-sparse matrix multiplication algorithm

To address the issues identified we have developed a semi-sparse matrix multiplication approach to avoid the redundant traverses on the selected vector in matrix multiplication. Each vector multiplication in the traditional SMO must traverse two sparse vectors. In our semi-sparse algorithm we initially make the selected vector (e.g., V_S) dense by creating a *dense*

array A_S^D to store both non-zero values in V_S and zero excluded in V_S. Next, the A_S^D is multiplied with all sparse vectors in the matrix sequentially. Fig. 1 illustrates the process of *densification* and multiplication of the selected vector V_S with a sparse vector V_i in the matrix.

Specifically, our semi-sparse algorithm works in two phases: (1) *selected vector densification*, and (2) *matrix multiplication*. In the *selected vector densification* phase, our algorithm first creates an array *wrk[]* in which all elements are initialized as 0; this is followed by a second stage in which all non-zero elements in V_S are filled in the corresponding locations of the wrk[]. In the *matrix multiplication* phase, the array wrk[] is multiplied with each sparse vector in the matrix.

The pseudo-code is described in Algorithm 2 where: N_S and N_i are the number of non-zero elements of the selected vector V_S and the trained vector V_i in the matrix respectively; N_V are the number of vectors in the trained matrix. Note that Algorithm 2 does not traverse the array wrk[]; it only locates index2[] and multiplies corresponding value2[] with elements of the wrk[].

Algorithm 2. Static semi-sparse matrix multiplication algorithm (in stack) (SSS-SMO)

```
1: i = 0, p1 = 0, p2 = 0, dot = 0;
2: double wrk[MAXLEN] = {0};
3: double result[N_V] = {0};
4: while (p1 < N_S) {   // make the selected vector dense
     wrk[index[p1]] = value[p1];
     p1++;}
5: while (i < N_V) {
6:    while (p2 < N_i) {   // get comparing result on semi-sparse vector
     dot+=wrk[index[p2]] * value[p2];
     p2++;}
7:    result[i]=dot;
8:    i++;}
```

The data set in the trained matrix is generally comprehensive and high-dimensional. Compared with Algorithm 1, it can be found that our Algorithm 2 significantly reduces the time taken to traverse and judge the selected vector V_S. Therefore, the time required for matrix multiplication will be significantly reduced because the selected vector V_S is traversed and judged only once during the whole matrix multiplication.

3.3. Performance analysis for our semi-sparse algorithm

For large-scale classification tasks, our semi-sparse algorithm can significantly speed up the training process. Let there be a sparse trained matrix $A_{m\times n}$. A selected vector V_S needs to multiply with n sparse vectors $V_i (1 \leqslant i \leqslant n)$ in the $A_{m\times n}$ during a matrix multiplication. The traditional SMO algorithm traverses and judges the selected vector V_S n times while our

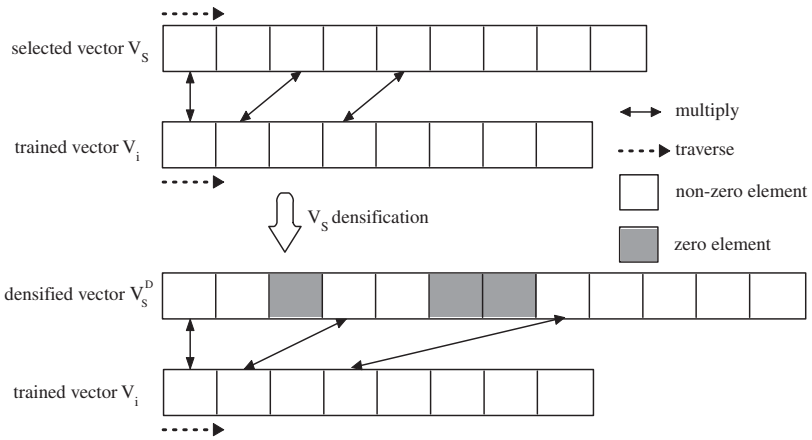

Fig. 1. Semi-sparse vector multiplication.

semi-sparse algorithm traverses the V_S only once. Specifically, in our semi-sparse algorithm, all vector multiplications are carried out directly on the dense array *wrk[]*. Consequently, for a vector multiplication between the V_S and a V_i, the traditional SMO algorithm will compare and judge $O(N_S + N_i)$ times while our semi-sparse algorithm only needs $O(N_i)$ times. This is the main reason why our semi-sparse algorithm can accelerate the matrix multiplication.

To multiply the selected vector V_S with the training matrix $A_{m\times n}$, the traditional SMO algorithm needs the time t_{SMO} to traverses vectors, which can be formulated as follows.

$$t_{SMO} = N_S \times n \times t_S + \sum_{i=1}^{n} N_i \times t_i \tag{1}$$

where t_S and t_i are the time to find one element in the V_S and the V_i respectively. For the same matrix multiplication, our semi-sparse algorithm needs the time t_{SS-SMO}, which can be calculated in the following formula.

$$t_{SS-SMO} = N_S \times t_S + \sum_{i=1}^{n} N_i \times t_i \tag{2}$$

Let there be a V_S and a matrix $A_{200\times 100}$. The V_S needs to multiply with the 100 sparse vectors in the $A_{200\times 100}$. Without losing generality, we assume all vectors have the same number of non-zero elements such that $N_S = N_i = N$ and the same traversing time such that $t_S = t_i = t$. In this case, the traditional SMO needs the time $t_{SMO} = 100 \times N_S \times t_S + \sum_{i=1}^{100} N_i \times t_i = 200N \times t$ while our semi-sparse algorithm requires the time $t_{SS-SMO} = N_S \times t_S + \sum_{i=1}^{100} N_i \times t_i = 101N \times t$, which is approximately 50% of that in the traditional SMO algorithm. Experimental results in Section 5 also verify this conclusion.

3.4. Dynamical implementation

Algorithm 3. Dynamical semi-sparse matrix multiplication algorithm (in heap) (DSS-SMO)

```
1: i = 0, p1 = 0, p2 = 0, dot = 0;
2: double *wrk = new double [M_S];
3: double result[N_V]=0;
4: memset (wrk,0,define*sizeof (double));
5: while (p1  < N_S) {   // make the selected p1 into a dense vector
     wrk[index[p1]] = value[p1];
     p1++;}
6: while (i  < N_V) {
7:    while (p2 < N_i) {   // get comparing result on semi-sparse vector
     dot+=wrk[index[p2]]*value[p2];
     p2++;}
8:    result[i]=dot;
9:    i++;}
```

In general, our Algorithm 2 can be implemented as a function or a method using C or C++ and when encoded in these programming languages the algorithm should run in a stack. A densified array wrk[] can be allocated storage space statically so that Algorithm 2 can run faster than other implementations.

However, Algorithm 2 needs an additional big dense array *wrk[]* to *densify* the selected vector V_S. For a selected vector V_S with m non-zero elements and M dimensions (generally speaking, $M \gg m$), Algorithm 2 requires (M-m) units of additional storage capacity as compared to the traditional SMO. What is more, the array *wrk[]* in Algorithm 2 should be allocated in terms of the largest data set, this results in wasted space to small training data sets. In the worst case scenario, the high-dimensional training matrix $A_{m\times n}$ could exceed the default capacity of a stack. If a static variable is set as 2 M by default in C++ compiling environment, for example, it can only contain a 0.5 M double array *wrk[]*.

Based on the above analysis, we have developed a further dynamic implementation of our novel semi-sparse approach. This is shown in Algorithm 3 where M_S is the dimension of the selected vector V_S and the *wrk[]* is a dynamic array in the heap area. In particular, the size of the dense array *wrk[]* can be dynamically specified as a parameter of the method or function when we choose different selected vectors from a training matrix. Although Algorithm 3 can solve the dynamic memory allocation issues for the dense array *wrk[]* in Algorithm 2, the initiation (set the dense vector *wrk* with zero elements memset (*wrk*, 0, $n \times$ sizeof (double))) of the dense array *wrk[]* will consume a little more time than that required in Algorithm 2.

4. Multi-class multi-label classification and parallelization for SVMTorch

This section considers the multi-label classification and parallelization for SVMTorch.

4.1. multi-class multi-label classification extension for SVMTorch

Text classification involves two basic conceptions: *class* and *label*. Let a data set S consist of K samples such that $S = \{S_i | 1 \leqslant i \leqslant K\}$ belong to M classes and N labels. We use S_i, S_i^c and $S_i^l (1 \leqslant i \leqslant K)$ to denote the ith sample, the class set of the S_i and the label set of the S_i. Further, $|S_i^c|$ and $|S_i^l| (1 \leqslant i \leqslant K)$ refer to the number of class(es) and label(s) in S_i. Generally, classification problems can be categorized as follows.

- *Two-class (TC) classification*. In TC classification, there are two classes in S and any sample $S_i \in S$ belongs to only one class such that $\forall S_i,\ |S_i^c| = 1$. In this case, any sample $S_i \in S$ can have only one label.
- *Multi-class single-label (MCSL) classification*. A *MCSL* data set has M ($M > 2$) classes, where any two classes are disjoint. Moreover, in the data set, there exists at least one sample $S_i (1 \leqslant i \leqslant K)$ that belongs to more than one class such that $\exists S_i,\ |S_i^c| \geqslant 2$. On the other hand, any sample S_i has exactly one label such that $\forall S_i,\ |S_i^l| = 1$.
- *Multi-class multi-label (MCML) classification*. In a *MCML* data set, at least one sample S_i belongs to more than one class such that $\exists S_i,\ |S_i^c| \geqslant 2$. At the same time, there are at least one sample S_i has more than one label such that $\exists S_j,\ |S_j^l| \geqslant 2$. Obviously, a data set in the *MCML* classification includes more than one classes and more than one labels.

4.1.1. Multi-class classification with single label or multiple labels

Preliminarily, the SVM is designed for only two-class classification tasks where each sample belongs to one of two classes: A and B. However, in 'real-world' applications there are generally multiple classes. For example, web pages may generally include many classes including: news, sports and education. Therefore, SVM based multi-class single label classification approaches have been well researched and are well documented in the literaturel.

The *TC* classification needs only one classifier which classifies each sample to the nearest class through feature matching. Multi-class classification is achieved mainly through two categories of method: decomposition. and a multi-class learning model. The front has faster training speed so that it is widely researched and has been frequently used. In general, decomposition based schemes use the following three policies: *one-against-one*, *one-against-many* and *output coding*. The *one-against-many* approach exhibits fast training speeds as well as high precision. The basic idea of this approach is to set up M 2-class classifiers for a M-class classification task. The ith 2-class classifier divides samples based on two-class classification, taking the sample S_i as a class and other samples as another class. For a *MCSL* classification, one-against-many use the M 2-class classifiers to directly divide each sample into a specific class and label. However, it is more complex for a MCML classification. Generally speaking, in the first step, the M 2-class classifiers transfer the MCML problem into multiple equivalent MCSL tasks. In the second step, the classifier first seeks a threshold, and decides which class and label each sample belongs to based on the threshold.

4.1.2. SVMTorch based multi-class multi-label classification

The SVM was originally designed for two-class classification and then extended to multi-class single-label classification. SVMTorch is a SMO based classification tool and while the SMTorch is effective in TC and MCSL classifications it cannot support the MCML classification. We extended the SVMTorch to enable MCML classification by the addition of a new class *MultiLableWorker*.

The key point of decomposition-based MCML classification is to transfer a MCML problem into multiple MCSL problems. For a *MCSL* data set with M classes, our *MultiLableWorker* initially constructs M two-class classifiers to transfer a MCML task into M MCSL tasks using its function *formLabel ()* as shown in Algorithm 4, and then matches each sample with labels through its function *formResult ()*, which is similar to the *formLabel ()*.

Traditional classification schemes use the two metrics: *recall* (r) and *precision* (p) to measure classification performance. They are formally described as the following formulas.

$$r = \frac{a}{a+c} \times 100\%$$

$$p = \frac{a}{a+b} \times 100\%$$

where a is the number of samples correctly classified as a specific class A; b refers to the number of samples that are mistakenly classified as the class A but belong to other class(es); c means the number of samples that belong to the class A but are mistakenly classified as other class(es). The metrics r and p are conflict to each other. Only considering the recall or the precision will eventually results in a large deviation. We combine these two metrics and propose the following metrics F_1 to evaluate classification performance of classifiers.

$$F_1 = \frac{2 \times p \times r}{p + r} \times 100\% \tag{3}$$

Algorithm 4. Transform a MCML problem into MCSL classification

```
Input: Training and testing samples
Output: Classified samples
1: formLabel (string file, real ** label, int &numExamples, int cl, int * fileID, int len) {
2:    ifstream f;
3:    f.open (file.c_str ());
4:    if (!f) error ();
      // get two-category information
5:    double * label_info  = newdouble[len];
6:    map <int, map<int,int>> fileID2Label;
7:    map <int,int> labels;
8:    int fid = 0, labelInfo = −1;
9:    string s = "";
10:    while (!f.eof ()) {
11:       f ≫ fid;       // get file ID
12:       labels = fileID2Label[fid];
13:       for (int i = 0; i < len; i++) {       // get label information of all samples in the file
             f ≫s;
             if (s == "#") break;
             labelInfo  = atoi (s.c_str ()) − 1;
             fileID2Label[fid][labelInfo] = 0;}}
      // transfer multi-label into two-category label
14:    numExamples = 0;
15:    for (int i = 0; i < len; i++){
          labels  = fileID2Label[fileID[i]];
          if (labels.count (cl) > 0){
             label_info[i] = 1.0;
             numExamples++;}
          else
             label_info[i] = −1.0;}
16:    labels.clear();
17:    fileID2Label.clear();
18:    f.close();
19:    * label = label_info;}
```

4.2. SVMTorch parallelization

The current SVMTorch cannot run in multi-core processor and cluster systems. With the training matrix growing exponentially larger, the long training time makes the SVMTorch inefficient even unusable to MUC environments. The Message Passing Interface (MPI) is a category of parallel programming language based on message communication. It is efficient for computation-intensive tasks without excessive data communication. In most cases, the SVMTorch needs many iterations for a training process; this represents a typical computationally-intensive application, especially for multi-classification and multi-label training. After analyzing the source code of the SVMTorch we identified that the SVMTorch executes the training and testing tasks sequentially. On the other hand, different 2-class classifiers in the SVMTorch execute independently so that these 2-class classifiers can work in parallel to speed up training and testing processes. Based on MPI, we have improved the *enhanced* SVMTorch by transplanting it to multi-core processor and cluster systems to speed up the training process.

Algorithm 5. Training parallelization

```
1: int worker_id, procs_num, src_p, dest, namelen, total_result;
2: char proc_name[MPI_MAX_PROCESSOR_NAME];
3: MPI::Init (argc, argv);
4: worker_id  = MPI::COMM_WORLD.Get_rank ();
5: procs_num  = MPI::COMM_WORLD.Get_size ();
6: MPI::Get_processor_name (proc_name, namelen);
7: for (int cl  = params.first_class; cl ⩽ params.last_class; cl++) {
8:    if (worker_id == (cl % procs_num)) {
        cout ≪"\n Training class" ≪ cl ≪"against the others \ n";
        if (params.multi_m) {
          //import 2-classification label information
9:         io.formLabel ("train_vector_class.txt", &y_temp, numExamples, cl, file_id, l);}
10:       else {//multi-label classification
          formLabelInfo (y, y_temp, l, numExamples, cl);
11:         cout ≪"# class" ≪cl # ≪"has "≪ numExamples ≪"examples ..."≪endl;
12:         formOutFile (file_out, argv[argc - 1], cl);
13:         bsvm (data, sdata, arr_size, y_temp, l, dim, &params, file_out,comment);}} // parallel task allocation
    ...;
    }
```

4.2.1. Parallel training

We parallelize SVMTorch through using MPI to distribute multiple 2-class classification sub-tasks on cores (or nodes). These concurrent execution units share a sparse training matrix. Parallel training is illustrated in Algorithm 5, which works in the following steps: including MPI head file, declaring parameters, initializing MPI run-time environment, classifying multi-class multi-label samples and finally ending MPI environment. During the training process, our algorithm provides a set of complete training data for each of parallel training processes to reduce messaging overhead. Communications among processors only involves execution results and execution marks. Our parallel approach only uses the following basic MPI functions to startup and exit MPI environments, recognize processes, and discover and receive messages.

- MPI_INIT( ): initialize MPI environment
- MPI_COMM_SIZE( ): query the number of processes
- MPI_COMM_RANK(): query the execution marks
- MPI_SEND(): send a message
- MPI_RECV(): receive a message
- MPI_FINALIZE: end the MPI environment

By the end of a training process, we save training results in a model file which is stored in a NFS based cluster. Moreover, we do not synchronize any intermediate result to keep different 2-class classifiers independently. Such the design further improves the training speed.

4.2.2. Parallel testing

Our parallelized SVMTorch synchronizes the classification process through saving final trained data in a file stored in NFS. Any execution unit can read all the training results during the testing process. Specifically, our scheme compares tested data sets with original labels, and organizes the tested results as a 8-dimension vector that mainly includes the number of samples classified mistakenly, the number of samples classified correctly, recall, precision and F_1. This vector will be transferred to a management process responsible for testing statistics. Our parallelization testing scheme is illustrated in Algorithm 6. We use Ready-mode based Point to Point communication and Group communication. The synchronization statement *MPI_Barrier (MPI_COMM_WORLD)* is used for the final statistics.

Algorithm 6. Testing parallelization

```
1:  for (int cl = 0; cl < n_class; cl++) {
2:    if (worker_id == (cl % procs_num)) {
3:      cout ≪"\ n testing class " ≪ cl ≪ endl;
4:      string file = argv[argc - 2];    // get the file type
        file = getStoreKernel (file, cl);
5:      real norm = svmTest (file, data, sdata, a_size, y_temp, l,&params);      // test the text vectors
6:      getBinStat (params.multi_m, cl, y_temp, l, stat, file_id);      // count testing results
7:      if (worker_id > 0) {// send statistic information to trace program
          MPI_Send (stat, 8, MPI_DOUBLE, 0, 123, MPI_COMM_WORLD);}
8:      else {
          getBinSVM (stat, res);}}      //send testing results to scheduling process manager
9:    if (worker_id == 0) {
        memset (stat, 0, 8);
        src_p = cl % procs_num;
        if (src_p > 0) {
          MPI_Recv (stat, 8, MPI_DOUBLE, src_p, 123, MPI_COMM_WORLD,&status);
          getBinSVM (stat, res);}}
10:   MPI_Barrier (MPI_COMM_WORLD);
      . . .;
    }
```

5. Experiments and performance evaluation

We have implemented our semi-sparse matrix multiplication algorithms, our multi-class multi-label classification approach, and parallelization scheme. In this section, we evaluate our solutions through comprehensive experiments and performance analysis. Specifically, we demonstrate the performance improvement derived using from our proposed solutions through comparing the following related schemes, which all rely on the same environment.

- The traditional SMO algorithm (SMO).
- Our semi-sparse matrix multiplication algorithm with static stack implementation (SSS-SMO, i.e., Algorithm 2).
- Our semi-sparse matrix multiplication algorithm with dynamical heap implementation (DSS-SMO, i.e., Algorithm 3).

5.1. Environment and setting

We implemented all solutions using C++ and ran them on a personal computer with 2.33 GHz Pentium CPU and 2G memory in Windows XP. In all the implementations, we use the Gaussian Kernel as a typical demonstration for our performance evaluations. In the text classification, the following two parameters have significant impact to classification results:

- *Error tolerance ratio* (simply marked as *e*). We vary ***e*** from 0.01 to 0.1 to evaluate the above three solutions. When we test how the performance change with ***c***, it is fixed as 0.01.
- *Trade-off between training error and training margin* (simply marked as **c**). In the experiments, we vary ***c*** from 200 to 2000. Its default value is set as 100.

We use *e* and *c* as parameters to evaluate the above three solutions. Each point in the following figures is derived from the average of 10 independent runs. In the testing the size of the dense array *wrk[]* is set as 50000 for our SSS-SMO algorithm because the maximal dimension in the training matrix is 31138.

5.2. Performance evaluation

5.2.1. Performance evaluation on our semi-sparse algorithm

In evaluating the performance we have utilized both the balanced corpus 20-newsgroups and the skewed corpus WebKB to demonstrate the performance of the three solutions under different scenarios.

- *WebKB*. This data set contains web pages gathered from computer science departments of universities. The pages involve seven categories: *student*, *faculty*, *staff*, *course*, *project*, *department* and *other*. WebKB contains 8203 pages (7031 training pages and 1172 testing pages). To distinguish impact from different parts in pages, we assign "Title", "H1" and "URL" parts five times of weight than that in "Body" part in web pages.

- *20-newsgroup*. This data set is generated from 20 Usenet newsgroups. It consists of 19,997 text messages, around one thousand text messages per category. Approximately 4% of the articles are cross-posted. We only keep "Subject", "Keywords" and "Content" parts in these articles and assign the "Subject" and "Keywords" parts five times of weight than that in "Contents".

We compare our semi-sparse algorithms (SSS-SMO and DSS-SMO) with the traditional SMO algorithm in terms of *training time* and *classification accuracy*.

Training time vs **e**. Our semi-sparse algorithms can significantly improve the training time under different error tolerance ratios *e*. Fig. 2 shows that both SSS-SMO and DSS-SMO significantly outperform over SMO in terms of training time when we use the skewed WebKB benchmark. In this case, our semi-sparse algorithms can reduce the training time to around 53.7% of that in the traditional SMO algorithm because the semi-sparse algorithm avoids the redundant traverses to the selected vector. Moreover, the SSS-SMO runs a little faster than the DSS-SMO. The reason is that the SSS-SMO is implemented on stack area so that it can allocate memory space more quickly than the DSS-SMO on heap area.

When the balanced 20-newsgroup was used as a benchmark, we observed a similar conclusion. As shown in Fig. 3, the training time is reduced to around 75% of that in the SMO. In this experiment, the static SSS-SMO still outperforms a little over the dynamic DSS-SMO.

In conclusion, our semi-sparse algorithm can significantly reduce the training time for both balanced and unbalanced data sets.

Training time vs **c**. In this experiment, we have tested how the training time changes with different values for *c*. As illustrated in Fig. 4, training time in our SSS-SMO and DSS-SMO has small growth on the skewed corpus WebKB as the parameter *c* increases. However, Fig. 5 reveals that the training time reduces a little with the increase values for the parameter *c* when we use the balanced corpus 20-newsgroup. The reverse variation tendency results from the different characteristics of the two used benchmarks. From Figs. 4 and 5, we find that our semi-sparse algorithms SSS-SMO and DSS-SMO always shows a significant reduce the training time when compared to the traditional SMO irrespective of whether we use balanced or unbalanced data sets. Another advantage of our SSS-SMO and DSS-SMO is that their training time is more stable than the traditional SMO algorithm under different **c**.

Classification accuracy vs **e**. This experiment tests how the classification accuracy in the three solutions changes with the different values for the parameter **e**; the results are shown in Figs. 6 and 7 where the WebKB and the 20-newsgroup are used as benchmarks respectively. From the two figures we can conclude that our SSS-SMO and DSS-SMO do not impact the classification accuracy and have been shown to improve accuracy in cases where they run on the unbalanced corpus. In fact, our semi-sparse algorithms only speed up matrix multiplication and do not impair the classification accuracy.

Classification accuracy vs **c**. Here, we set the parameter ***e*** such that $e = 0.01$ and examined how the classification accuracy changes with the different parametric values for *c*. Figs. 8 and 9 show that our SSS-SMO and DSS-SMO have the same classification accuracy as the traditional SMO algorithm.

5.2.2. Performance evaluation on our multi-class multi-label classification approach

The traditional SVMTorch only supports multi-class single-label classification. We extend the traditional SVMTorch to enable classification of multi-class multi-label data sets. In this experiment, we use the unbalanced corpus *Reuters-21578*, which is widely used as a multi-label classification benchmark, to examine our multi-class multi-label classification approach. The Reuters-21578 consists of 21578 samples with 135 classes. We have selected 7723 multi-label samples from

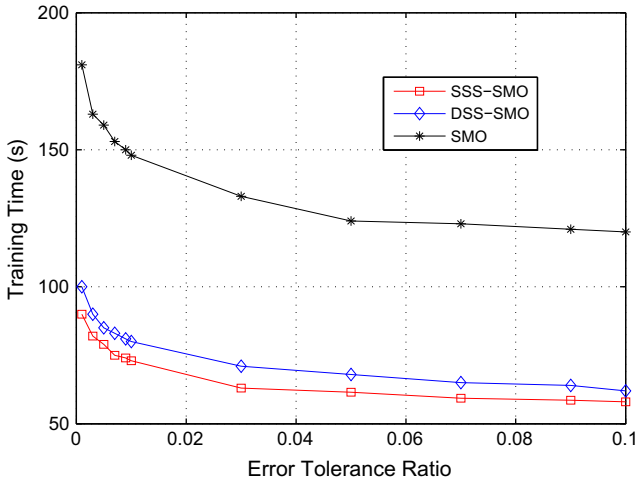

Fig. 2. Training time for the WebKB under different error tolerance ratio *e*.

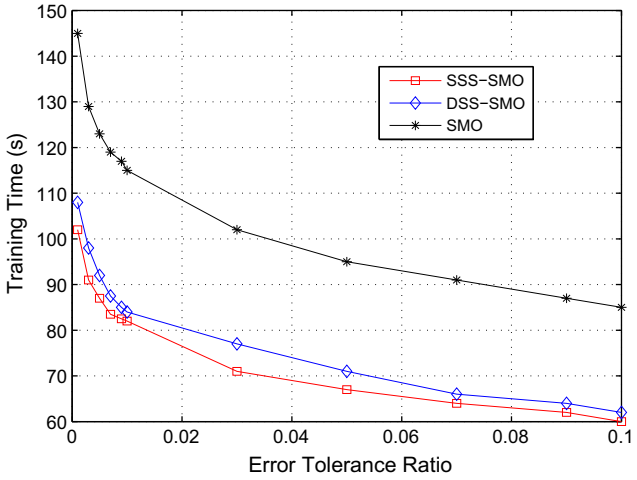

Fig. 3. Training time for the 20-newsgroup under different error tolerance ratio *e*.

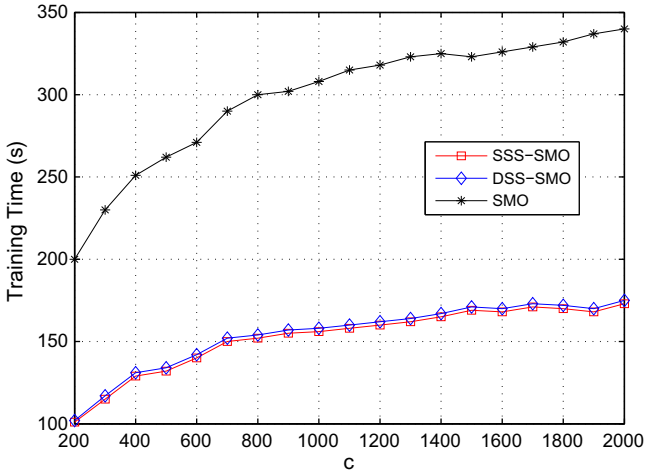

Fig. 4. Training time for the WebKB under different parameter *c*.

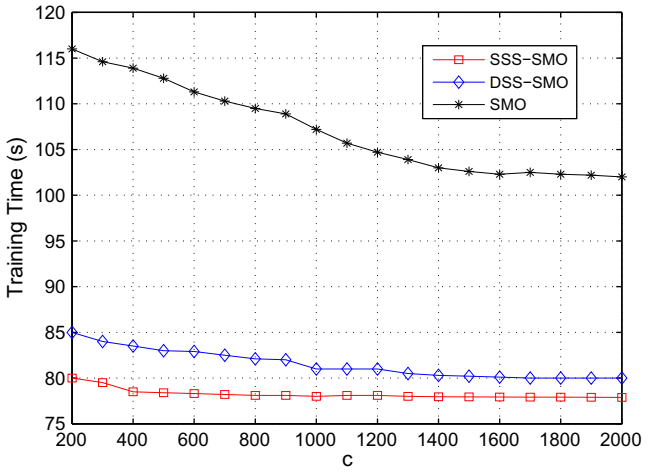

Fig. 5. Training time for the 20-newsgroup under different parameter *c*.

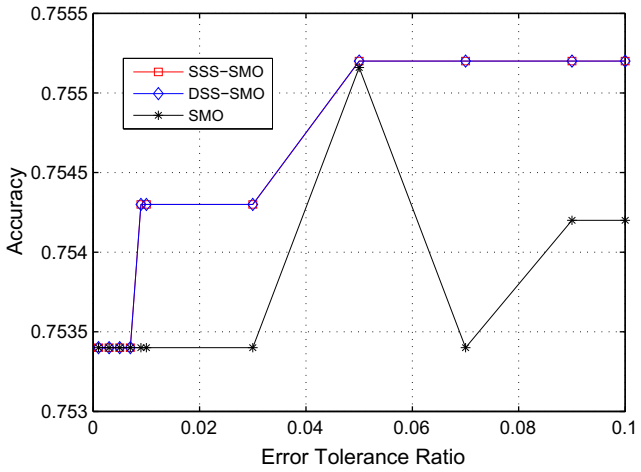

Fig. 6. Classification accuracy for the WebKB with the parameter *e*.

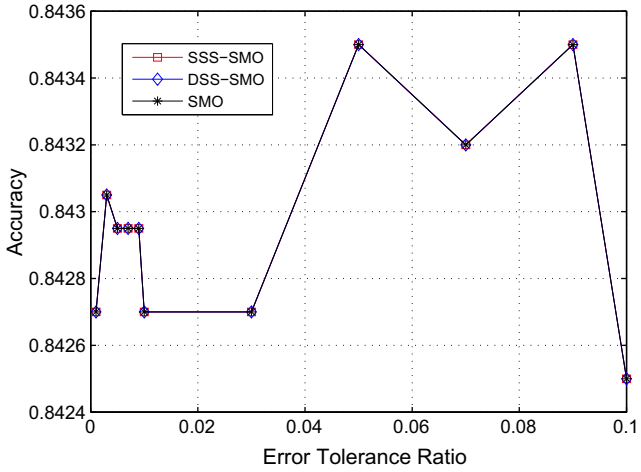

Fig. 7. Classification accuracy for the 20-newsgroup with the parameter *e*.

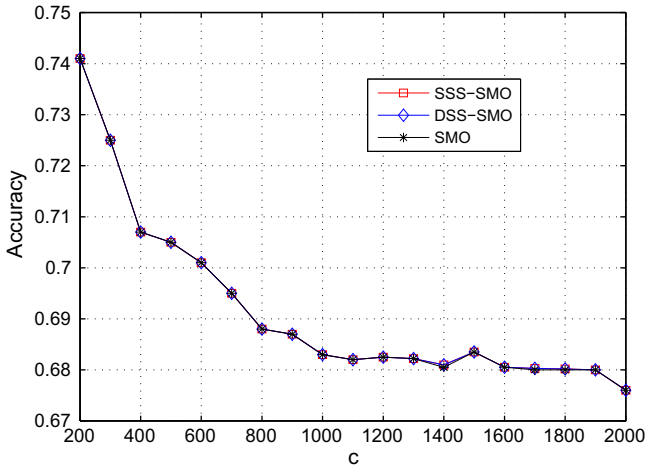

Fig. 8. Classification accuracy for the WebKB with the parameter *c*.

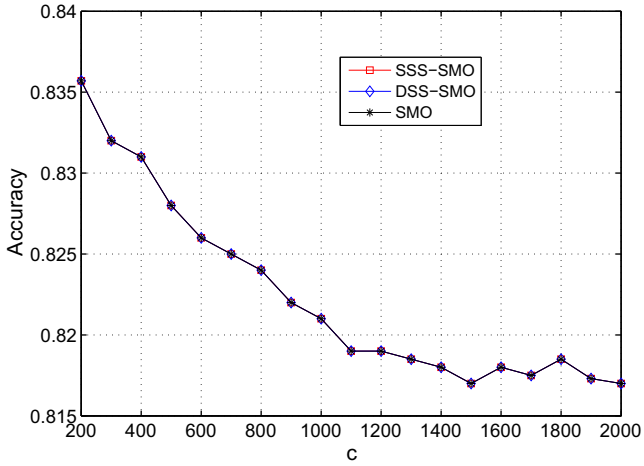

Fig. 9. Classification accuracy for the 20-newsgroup with the parameter *c*.

Table 1
Results of our multi-class multi-label classifier on the Reuters-21578.

Class	Tested Doc (a + c)	Total Doc (a + b)	Doc classified correctly (a)	Other Doc (c)	Doc classified mistakenly (b)	Recall (%)	Precision (%)	F_1 (%)
Earn	1,080	1,047	1,035	45	12	95.83	98.85	97.32
Acq	718	678	646	72	32	89.97	95.28	92.55
Crude	186	161	139	47	22	74.73	86.34	80.12
Money-fx	179	168	117	62	51	65.36	69.64	67.44
Grain	148	135	120	28	15	81.08	88.89	84.81
Interest	131	116	89	42	27	67.94	76.72	72.06
Trade	116	114	79	37	35	68.1	69.3	68.7
Ship	87	52	44	43	8	50.57	84.62	63.31
Wheat	71	59	47	24	12	66.2	79.66	72.31
Corn	56	52	40	16	12	71.43	76.92	74.07

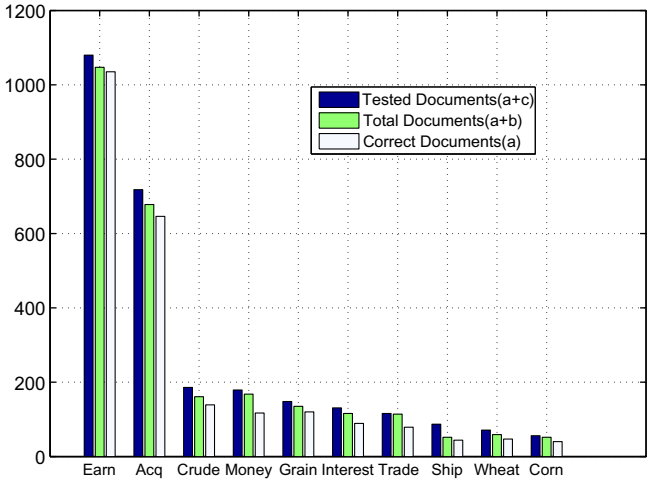

Fig. 10. The number of documents classified by our multi-class multi-label classification approach.

its training set where these training samples belong to 90 classes. We classified all the training samples and found that most classes in the Reuters-21578 include only a relatively few samples. Therefore, we have counted the first 10 classes: *Earn*, *Acq*, *Crude*, *Money-Fx*, *Grain*, *Interest*, *Trade*, *Ship*, *Wheat* and *Corn* and have listed their results in Table 1. Figs. 10 and 11 intuitively show that our multi-class multi-label classification approach can work well.

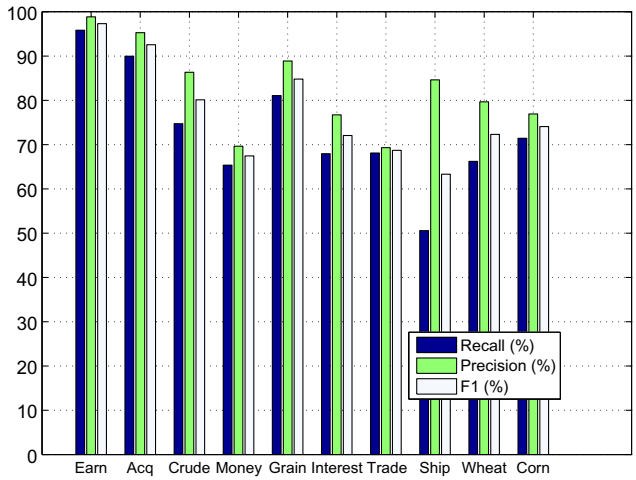

Fig. 11. The classification performance of our multi-class multi-label classification approach.

Table 2
Classification results of our Chinese web page classifier.

Class	Tested Doc (a + c)	Total Doc (a + b)	Doc classified correctly (a)	Other Doc (c)	Doc classified mistakenly (b)	Recall (%)	Precision (%)
Traffic	71	68	68	3	0	95.78	100
Sport	149	153	149	0	4	100	97.39
Military	83	77	66	17	11	79.52	85.71
Medicine	68	67	65	3	2	97.06	98.51
Politics	167	181	160	7	21	95.81	88.4
Education	73	71	68	5	3	93.15	95.78
Environment	67	61	59	8	2	88.06	96.72
Economy	108	107	103	5	4	95.37	96.26
Art	82	83	81	1	2	98.78	97.59
Computer	66	66	64	2	2	96.7	96.7

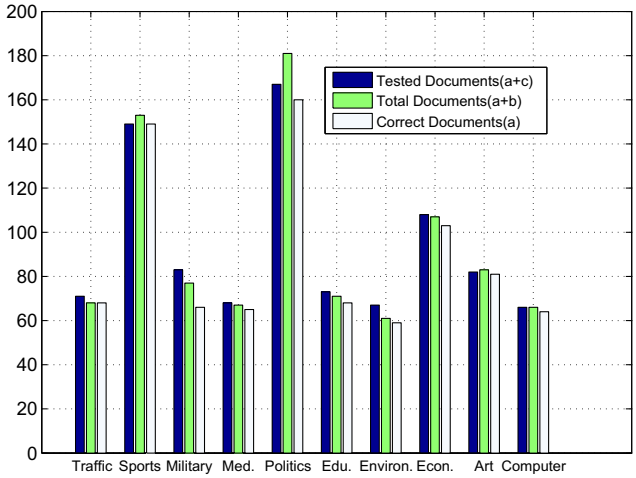

Fig. 12. The number of classified documents.

5.2.3. Performance evaluation on our parallelization approach

We have parallelized SVMTorch, which has been improved using our solution, on a cluster with 10 nodes. Each node has eight 2.0 Hz cores and a 8G memory. We install centos 5.4 in the cluster, use NIS to support the Single Sign-On, and manage

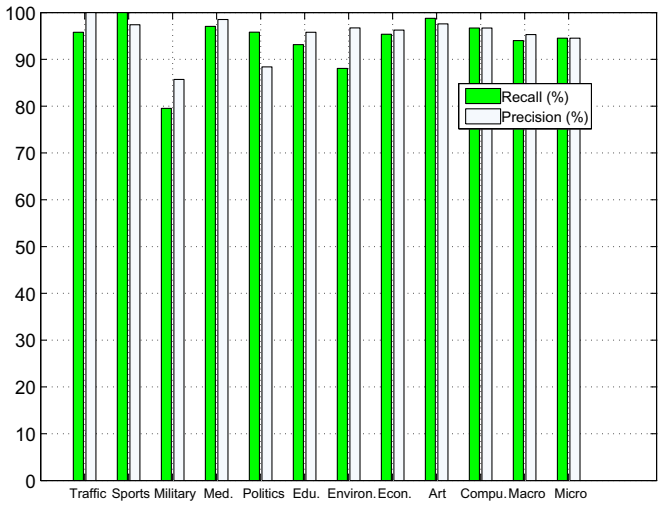

Fig. 13. Classification performance of our Chinese web page classifier.

files through the NFS. We have run our improved SVMTorch to train the 20-newsgroups; the training time is 10, 18 and 143s when 10 nodes are used, only one node and only one core respectively. This result reveals that our parallelization scheme significantly reduces the training time; the greater the number of cores used in parallel in a classification task the greater the improvement in performance achieved by our parallelization approach.

By comparison, the time taken in training the WebKB in the cluster is 67 s; the time taken is noticeably greater than for the 20-newsgroup. The rationale for this is that the WebKB is unbalanced. In fact, the third class in the WebKB has almost 50% samples. Therefore the time spent in training the third-class samples is much greater which results increased training time than for other classes; the result is that other parts have to wait for the completion of the training for the third class. Future projected work will target further improvements in parallelization performance for unbalanced data classification through super thread idea.

5.2.4. A Chinese web page classifier and performance evaluation

It is very important to classify multi-class multi-label web pages automatically and efficiently as the WWW has become ubiquitous and pervasive with the development of mobile wireless technologies. We have developed a Chinese web page classifier based on our semi-sparse algorithm using the multi-label classification and parallelization approaches. The classifier works in the training and the testing phases. In this section, we focus on evaluating the classification performance of our SVMTorch-based classifier.

Classifier testing has employed the FDU corpus with 19437 files: 9833 files for training and 9604 files for testing. These files belongs multiple classes and multiple labels, e.g., environment, computer, traffic, education, economy, military, medicine, art and politics. Our classifier pay "Title", "H1" and "URL" parts 5 times of weight than "body" area.

In this part, we use not only *recall* and *precision* but also *macro averaging* ($\bar{r}$ and $\bar{p}$) and *micro averaging* ($\tilde{r}$ and $\tilde{p}$) as the performance metrics to clearly examine the classification ability of our classifier.

$$\bar{r} = \frac{\sum_{s\in S} r_s}{|S|}; \quad \bar{p} = \frac{\sum_{s\in S} p_s}{|S|} \tag{4}$$

$$\tilde{r} = \frac{\sum_{s\in S} a}{\sum_{s\in S} a + \sum_{s\in S} c}; \quad \tilde{p} = \frac{\sum_{s\in S} a}{\sum_{s\in S} a + \sum_{s\in S} b} \tag{5}$$

where (1) r_s and p_s are the recall and precision of the sample s, (2) $\bar{r}$ and $\bar{p}$ are the *macro averaging recall* and *macro averaging precision*, (3) $\tilde{r}$ and $\tilde{p}$ are the *micro averaging recall* and *micro averaging precision*, (4) S is the set of samples, and (5) $|S|$ is the number of the samples. Moreover, a, b and c are the same as described above. Classification results are listed in Table 2. Figs. 12 and 13 show our classifier is sufficiently efficient to enable effective classification of Chinese web pages.

6. Conclusions

Context classification lies at the center of user-centric MUC where the aim is the provision of personalized service provision based on user preferences in dynamic mobile scenarios. To handle large volumes of contextual information and to meet the high demands of ubiquitous users, context classification for MUC environments has to be sufficiently fast, effective, and efficient. The SMO based SVMTorch, while widely used in the text classification, is inefficient for MUC context analysis in

terms of both low classification speed caused by inefficient matrix multiplication and its inherent inability to handle multi-label data classification.

To provide an effective basis upon which versatility and performance improvements in the classification of context information in large-scale MUC environments can be realized, we have proposed a novel classification approach which incorporates the following advantages:

- Our semi-sparse algorithm reduces the matrix multiplication time to around 50% of that in the traditional SMO algorithm.
- We extend the traditional SVMTorch to classify multi-class multi-label data sets.
- We parallelize SVMTorch, which is improved as discussed in this paper, to multi-core processor and cluster systems. It significantly speeds up the training and testing processes, especially for the balanced data sets.
- Our Chinese web page classifier can be applied to real web page classification.

The proposed solution presented in this paper has been evaluated and verified based on systematic experiments together with a rigorous theoretic analysis. In conclusion, the results demonstrate that the improvements achieved using our solution are proportional to the size of the training and testing data sets. Thus, the proposed solution is posited as an effective solution to context classification in large scale system in 'real-world' applications.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (NSFC) with Grant Nos. 61073148 and 61272442, and Key Basic Research Project of the Science and Technology Commission of Shanghai Municipality with Grant No. 12JC1405400.

References

[1] G.S. Aoude, V.R. Desaraju, L.H. Stephens, J.P. How, Driver behavior classification at intersections and validation on large naturalistic data set, IEEE Transactions on Intelligent Transportation Systems 13 (2) (2012) 724–736.
[2] E.J. Bayro-Corrochano, N. Arana-Daniel, Clifford support vector machines for classification, regression, and recurrence, IEEE Transactions on Neural Networks 21 (11) (2010) 1731–1746.
[3] K. Bernard, Y. Tarabalka, J. Angulo, et al, Spectral–spatial classification of hyperspectral data based on a stochastic minimum spanning forest approach, IEEE Transactions on Image Processing 21 (4) (2012) 2008–2021.
[4] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press, Pittsburgh, PA, 1992, pp. l44–l52.
[5] S. Bouyuklieva, I. Bouyukliev, An algorithm for classification of binary self-dual codes, IEEE Transactions on Information Theory 58 (6) (2012) 3933–3940.
[6] F. Bovolo, L. Bruzzone, L. Carlin, A novel technique for subpixel image classification based on support vector machine, IEEE Transactions on Image Processing 19 (11) (2010) 2983–2999.
[7] A. Bremler-Barr, D. Hendler, Space-efficient TCAM-based classification using gray coding, IEEE Transactions on Computers 61 (1) (2012) 18–30.
[8] C.J.C. Burges, B. Scholkopf, Improving the accuracy and speed of support vector learning machines, in: Advances in Neural Information Processing Systems 9, MIT Press, Cambridge, MA, 1997, pp. 375–381.
[9] L.J. Cao, S.S. Keerthi, C.J. Ong, J.Q. Zhang, et al, Parallel sequential minimal optimization for the training of support vector machines, IEEE Transactions on Neural Networks 17 (4) (2006) 1039–1049.
[10] S. Cesare, Y. Xiang, W.L. Zhou, Malwise – an effective and efficient classification system for packed and polymorphic malware, IEEE Transactions on Computer PP (99) (2012).
[11] X. Chang, S.C. Cheung, W.K. Chan, et al., Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications, in: Proceedigns of ICDCS, 2008, pp. 713–721.
[12] W.A. Chaovalitwongse, Y.J. Fan, Support feature machine for classification of abnormal brain activity, in: Proceedings of KDD 2007, August 2007, San Jose, California, USA, 2007.
[13] W.-J. Choi, T.-S. Choi, Genetic programming-based feature transform and classification for the automatic detection of pulmonary nodules on computed tomography images, Information Sciences 212 (1) (2012) 57–78.
[14] J.-F. Connolly, E. Granger, R. Sabourin, An adaptive classification system for video-based face recognition, Information Sciences 192 (1) (2012) 50–70.
[15] D. DeCoste, K. Wagstaff, Alpha seeding for support vector machines, in: Proceedings of International Conference on Knowledge Discovery and Data Mining(KDD-2000), 2000.
[16] Y.S. Dong, J.W. Ma, Bayesian texture classification based on contourlet transform and BYY harmony learning of poisson mixtures, IEEE Transactions on Image Processing 21 (3) (2012) 909–918.
[17] R. Hosseini, S.D. Qanadli, S. Barman, et al, An automatic approach for learning and tuning Gaussian interval type-2 fuzzy membership functions applied to lung CAD classification system, IEEE Transactions on Fuzzy Systems 20 (2) (2012) 224–234.
[18] C.W. Hsu, C.J. Lin, A simple decomposition method for support vector machines, Machines Learning 46 (2002) 291–314.
[19] D. Isa, L.H. Lee, V.P. Kallimani, R. RajKumar, Text document preprocessing with the Bayes formula for classification using the support vector machine, IEEE Transactions on Knowledge and Data Engineering 20 (9) (2008) 1264–1272.
[20] Jayadeva, R. Khemchandani, S. Chandra, Twin support vector machines for pattern classification, IEEE Transactions on Patter Analysis and Machine Intelligence 29 (5) (2007) 905–910.
[21] T. Joachims, Making large-scale support vector machine learning practical, Advances in Kernel Methods Support Vector Machines, 1999.
[22] T. Joachims. Web page on SVMLight: <http://www-ai.cs.uni-dortmund.de/SOFTWATE/SVMLIGHT/svmlight.eng.html>.
[23] A.J. Joshi, F. Porikli, N. Papanikolopoulos, Scalable active learning for multi-class image classification, in: IEEE Transactions on Pattern Analysis and Machine Intelligence PP(99), 2012, p. 1.
[24] T.G. Kanter, HotTown, enabling context-aware and extensible mobile interactive spaces, IEEE Wireless Communications 9 (5) (2002) 18–27.
[25] T.-W. Kuan, J.-F. Wang, J.-C. Wang, et al., VLSI design of an SVM learning core on sequential minimal optimization algorithm, in: IEEE Transactions on Very Large Scale Integration (VLSI) Systems PP(99), 2011, pp. 1–11.

[26] D.C. Li, C.W. Liu, Extending attribute information for small data set classification, IEEE Transactions on Knowledge and Data Engineering 24 (3) (2012) 452–464.
[27] C.T. Lin, C.M. Yeh, S.F. Liang, et al, Support-vector-based fuzzy neural network for pattern classification, IEEE Transactions on Fuzzy Systems 14 (1) (2006) 31–41.
[28] P. Lingras, C. Butz, Rough set based 1-v-1 and 1-v-r approaches to support vector machine multi-classification, Information Sciences 177 (2007) 3782–3798.
[29] J.M. Ma, M.N. Nguyen, J.C. Rajapakse, Gene classification using codon usage and support vector machines, IEEE/ACM Transactions on Computational Biology and Bioinformatics 6 (1) (2009) 134–143.
[30] S. Maji, A.C. Berg, J. Malik, Efficient classification for additive kernel SVMs, IEEE Transactions on Pattern Analysis and Machine Intelligence, PP (99) (2012).
[31] S. Maldonado, R. Weber, J. Basak, Simultaneous feature selection and classification using kernel-penalized support vector machines, Information Sciences 181 (2011) 115–128.
[32] M.M. Masud, Q. Chen, L. Khan, et al., Classification and adaptive novel class detection of feature-evolving data streams. IEEE Transactions on Knowledge and Data Engineering, PP (99) (2012).
[33] P.D. Meo, A. Nocera, G. Terracina, D. Ursino, Recommendation of similar users, resources and social networks in a social internetworking scenario, Information Sciences 181 (7) (2011) 1285–1305.
[34] S. Moustakidis, G. Mallinis, N. Koutsias, et al, SVM-Based Fuzzy Decision Trees for Classification of High Spatial Resolution Remote Sensing Images, IEEE Transactions on Geoscience and Remote Sensing 50 (1) (2012) 149–169.
[35] S. Mukherjee, E. Osuna, F. Girosi, Nonlinear prediction of chaotic time series using support vector machines, in: Proc. of IEEE NNSP'97, Amelia Island, FL, 1997.
[36] E. Osuna, R. Freund, F. Girosi, An improved training algorithm for support vector machines, in: Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing, New York, 1997, pp. 276–285.
[37] E. Osuna, R. Freund, F. Girosi, Training support vector machines: an application to face detection, in: Proceedings of IEEE Conference on Computer Vision and, Pattern Recognition, 1997, pp. 130–136.
[38] S.N. Pang, T. Ban, Y. Kadobayashi, N. Kasabov, Personalized mode transductive spanning SVM classification tree, Information Sciences 181 (2011) 2071–2085.
[39] M. Papadonikolakis, C.-S. Bouganis, Novel cascade FPGA accelerator for support vector machines classification, IEEE Transactions on Neural Networks and Learning Systems 23 (7) (2012) 1040–1052.
[40] X.J. Peng, D. Xu, Twin Mahalanobis distance-based suppor tvector machines for pattern recognition, Information Sciences 200 (1) (2012) 22–37.
[41] J.C. Platt, Fast Training of Support Vector Machines Using Sequential Minimal Potimization, Advances in Kernel Methods: Support Vector Machines (Book), MIT Press Cambridge, MA, USA, 1999.
[42] R.C. Prati, G.E.A.P.A. Batista, M.C. Monard, A survey on graphical methods for classification predictive performance evaluation, IEEE Transactions on Knowledge and Data Engineering 23 (11) (2011) 1601–1618.
[43] V. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.
[44] V. Viswanathana, K. Ilangob, Ranking semantic relationships between two entities using personalization in context specification, Information Sciences 207 (10) (2012) 35–49.
[45] J. Wu, A framework for learning comprehensible theories in XML document classification, IEEE Transactions on Knowledge and Data Engineering 24 (1) (2012) 1–14.
[46] Z.W. Yu, X.S. Zhou, D.Q. Zhang, et al, Supporting context-aware media recommendations for smart phones, IEEE Pervasive Computing 5 (3) (2006) 68–75.