Chapter 1

A Parallel and Modular Pattern Classification Framework for Large-Scale Problems

Bao-liang Lu[1,2] and Xiao-lin Wang[1]

[1]Center for Brain-Like Computing and Machine Intelligence
Department of Computer Science and Engineering
[2]MOE-Microsoft Key Laboratory for Intelligent Computing and Intelligent Systems
Shanghai Jiao Tong University
800 Dong Chuan Rd., Shanghai 200240, China
bllu@sjtu.edu.cn

The number of samples that are available on the internet to train pattern classifiers is increasing rapidly, while traditional pattern classification techniques based on a single computer system are powerless to process these large-scale data sets. This chapter presents a parallel and modular pattern classification framework for coping with large-scale pattern classification problems. The proposed framework follows a divide-and-conquer strategy that easily assigns a given large-scale problem to an available parallel and distributed computing infrastructure. The framework consists of three independent parts: decomposing training data sets, training component classifiers in a parallel way, and combining trained component classifiers. In order to evaluate the performance of the proposed framework, we perform experiments on a large-scale Japanese patent classification problem, containing about 3,500,000 patent documents. The experimental results show that our framework has the following attractive features: (a) The framework is general, and therefore any traditional pattern classification techniques such as support vector machines can be easily embedded in the framework as component classifiers. (b) The framework can incorporate explicit domain or prior knowledge into learning through the process of dividing training data sets. (c) The framework has good scalability and is easily implementable in hardware.

1 Introduction

More and more large-scale classification problems appear in the field of machine learning and pattern recognition. By large-scale we mean that the training set is extremely large, typically to such an extent that traditional classifiers cannot finish the learning task within a bearable time. For example, recently, the training of support vector machines (SVMs) on large real-world data sets is reported to take several weeks. With the emergence of these large-scale classification problems, the efficiency of classifiers, besides accuracy, has become a subject of great focus.

A large-scale classification problem has huge time and memory costs. The ultimate solution to reducing time and memory costs is parallelism, that is, splitting one large-scale classification task among several CPUs, by which the system can meet an expected efficiency requirement. Unfortunately, most traditional classifiers are sequential in spirit, so we have to expend great efforts in parallelizing them. Recently, several parallel implementations of SVMs have been released [8, 4, 14, 34, 26, 13]. There are also parallel implementations of other classifiers, such as k-NN [3].

A very useful technology for these large-scale classification problems is ensemble learning. It employs a group of classifiers for a classification task, instead of one single classifier. Ensemble learning aims to achieve high accuracy, and it shows great efficiency in large-scale problems. Some of the ensemble learning methods require the sub-classifiers to be trained sequentially, while the others allow parallel training sub-classifiers. For methods such as Adaboost, the training set of a sub-classifier relies on feedback from previous trained sub-classifiers, and thus parallel learning is infeasible. As for other parallel ensemble learning methods, they are ideal solutions for large-scale tasks.

In this chapter, we introduce a parallel and modular pattern classification framework, named min-max modular network (M^3-network) [20, 21, 24]. This framework is a kind of a parallel ensemble learning method. First we train component classifiers on the subsets of the original data set. Then we make these component classifiers produce predictions on the arriving test sample. In the end we integrate all the predictions by two module combination rules, namely the minimization principle and the maximization principle.

To solve a large-scale multi-class classification problem, the M^3-network consists of three main steps: (a) Decompose the original multi-class problem into two-class problems and further divide the two-class problems which are difficult to be learned into a series of relatively smaller and balanced two-class subproblems. (b) Train all the two-class subproblems in parallel. (c) Combine all the trained component classifiers into a hierarchical, parallel, and modular network that serves as the solution to the original problem.

Many studies prove that M^3-network is an efficient classifier, especially in solving large-scale and complex pattern classification problems [22, 10, 32, 19, 5, 6]. M^3-netowrk has the following three advantages over traditional pattern classification approaches: (a) A large-scale and complex multi-class problem can be decomposed to two-class subproblems as small as the user expects. (b) The two-class subproblems are independent to each other, therefore, they can be solved in parallel without the trouble of communication. (c) The two module combination principles are simple, which are easily implementable in software and hardware.

2 Min-Max Modular Network

Lu and Ito proposed the Min-Max Modular Network (M^3-network for short) in order to solve hard classification problems in 1997 [20, 21]. M^3-network is a sort

of ensemble learning methods that employs a group of classifiers for one classification problem. The M^3-network follows the principle of divide and conquer, dividing the whole problem into small pieces and solving them one by one, and adopts a 3-step work flow: task decomposition, training component classifiers and module combination (see Figure 1).

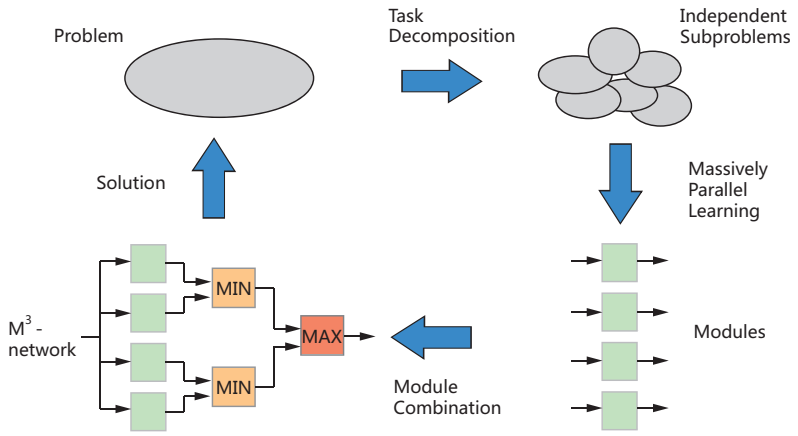


Fig. 1. Working Flow of the M^3-network

Before formally describing the M^3-network, we would like to present an illustration (Figure 2) [25]. Suppose that subfigure (a) is the two-class classification problem that needs to be solved, where small red disks represent positive samples and small blue rectangles represent negative ones. Though simple at first sight, it's actually a non-linearly separable problem. To demonstrate M^3-network learning, we divide samples of each class into two subsets, surrounded by dashed lines in subfigure (a). According to this partition of the training data set, we generate four smaller classification subproblems shown in subfigures (b) to (e). Then we train classifiers on these subproblems, where dashed lines represent the discriminant surfaces. After that we derive separately subfigures (f) and (g) from subfigures (b) through (e) with the *minimization* operator, resulting in negative zone expanding. Finally, we derive subfigure (h) from subfigures (f) and (g) with the *maximization* operator, resulting in positive zone expanding. From subfigure (h), it is obvious that we find the correct discriminant plane to the classification problem.

2.1 Task Decomposition

Let T denote the training data set for a K-class classification problem,

$$\mathcal{T} = \{(X_i, Y_i)\}_{i=1}^{L} \tag{1}$$

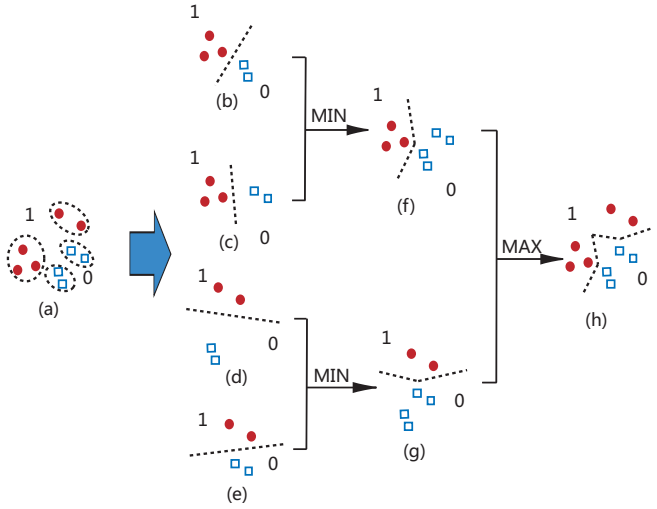


Fig. 2. Illustration of the M^3-network learning

where $X_i \in R^d$ is the training input, $Y_i \in R^K$ is the corresponding desired output, and L is the total number of training samples.

At the beginning of the task decomposition process, a large-scale K-class classification problem is divided into a series of relatively smaller two-class subproblems by using a one-versus-one or one-versus-rest task decomposition strategy. Suppose a K-class classification problem is decomposed by a one-versus-one strategy into the following $K(K-1)/2$ two-class subproblems:

$$\mathcal{T}_{ij} = \{(X_l^i, +1)\}_{l=1}^{L_i} \cup \{(X_l^j, -1)\}_{l=1}^{L_j} \quad \text{for } i = 1, \cdots, K \text{ and } j = i+1, \cdots, K \tag{2}$$

where $X_l^i \in \mathcal{X}_i$ and $X_l^j \in \mathcal{X}_j$ are the training inputs belonging to class $\mathcal{C}_i$ and class $\mathcal{C}_j$, respectively; $\mathcal{X}_i$ is the set of training inputs belonging to class $\mathcal{C}_i$; L_i denotes the number of data in $\mathcal{X}_i$; $\cup_{i=1}^K \mathcal{X}_i = \mathcal{X}$; and $\sum_{i=1}^K L_i = L$. In this chapter, the training data in a two-class problem are called *positive* training data if their desired outputs are $+1$, and they are called *negative* training data if their desired outputs are -1.

Even though the two-class problems defined by Eq. (2) are smaller than the original K-class problem, this partition may be not adequate for parallel learning. Since the number of samples in each class varies largely, the sizes of these two-class problems can be quite different. Thus the training period can be delayed by some larger two-class problems. A large class and a small class together form an imbalanced classification problem; there are too many samples on one side, which increases the difficulty in training classifiers [2]. To speed up training and to improve classification accuracy, all the large and imbalanced

two-class problems are further divided into smaller and more balanced two-class problems.

Assume that $\mathcal{X}_i$ is partitioned into N_i subsets in the form

$$\mathcal{X}_{ij} = \{X_l^{ij}\}_{l=1}^{L_i^j} \quad \text{for } j = 1, \cdots, N_i \text{ and } i = 1, \cdots, K, \tag{3}$$

where $1 \leq N_i \leq L_i$ and $\cup_{j=1}^{Ni} \mathcal{X}_{ij} = \mathcal{X}_i$.

After partitioning $\mathcal{X}_i$ into N_i subsets, every two-class problem $\mathcal{T}_{ij}$ defined by Eq. (2) is further divided into $N_i \times N_j$ smaller and more balanced two-class subproblems:

$$\mathcal{T}_{ij}^{(u,\,v)} = \{(X_l^{(iu)},\ +1)\}_{l=1}^{L_i^{(u)}} \cup \{(X_l^{(jv)},\ -1)\}_{l=1}^{L_j^{(v)}} \quad \text{for } u = 1, \cdots, N_i,\ v = 1, \cdots, N_j, \ i = 1, \cdots, K, \text{ and } j = i+1, \cdots, K, \tag{4}$$

where $X_l^{(iu)} \in \mathcal{X}_{iu}$ and $X_l^{(jv)} \in \mathcal{X}_{jv}$ are the training inputs belonging to class $\mathcal{C}_i$ and class $\mathcal{C}_j$, respectively; $\sum_{u=1}^{N_i} L_i^{(u)} = L_i$ and $\sum_{v=1}^{N_j} L_j^{(v)} = L_j$.

2.2 Training Component Classifiers

After task decomposition, all of the two-class subproblems are treated as completely independent, non-communicating tasks in the learning phase. Therefore, all the two-class subproblems defined by Eq. (4) are efficiently learned in a massively parallel way.

From Eqs. (2) and (4), we see that a K-class problem is divided into

$$\sum_{i=1}^{K-1} \sum_{j=i+1}^{K} N_i \times N_j \tag{5}$$

two-class subproblems. The number of training data for each of the two-class subproblems is approximately

$$\lceil L_i/N_i \rceil + \lceil L_j/N_j \rceil, \tag{6}$$

where $\lceil z \rceil$ denotes the smallest integer than or equal to z. Since $\lceil L_i/N_i \rceil + \lceil L_j/N_j \rceil$ is independent of the number of classes, K, the size of each two-class subproblems is much smaller than the original K-class problem for the reasonable values of N_i and N_j.

Traditional machine learning algorithms and pattern classification approaches such as multilayer neural networks [21], support vector machines [23] and k-nearest neighbor algorithm [35], can be used to learn the two-class subproblems defined by Eq. (4). In this chapter, most commonly used SVMs are selected as component classifiers and used to learn all of the two-class subproblems.

2.3 Module Combination

After all of the two-class subproblems defined by Eq. (4) have been learned by SVMs, all the trained SVMs are integrated into a M^3-SVM with the Min and Max units according to the minimization principle and maximization principle [23]. The function of the Min unit is to find a minimum value from its multiple inputs. The transfer function of the Min unit is given by

$$q(x) = \min_{i=1}^{P} \mathrm{M}_i(x) \tag{7}$$

where x denotes the input variable. The function of the Max unit is to find a maximum value from its multiple inputs. The transfer function of the Max unit is given by

$$q(x) = \max_{i=1}^{P} \mathrm{M}_i(x) \tag{8}$$

Minimization Principle : Suppose a two-class problem $\mathcal{B}$ is divided into P smaller two-class subproblems, $\mathcal{B}_i$ for $i = 1, \cdots, P$, and also suppose that all the two-class subproblems have the same positive training data and different negative training data. If the P two-class subproblems are correctly learned by the corresponding P individual SVMs, M_i for $i = 1, \cdots, P$, then the combination of the P trained SVMs with a Min unit will produce the correct output for all the training inputs in $\mathcal{B}$.

Maximization Principle: Suppose a two-class problem $\mathcal{B}$ is divided into P smaller two-class subproblems, $\mathcal{B}_i$ for $i = 1, \cdots, P$, and also suppose that all the two-class subproblems have the same negative training data and different positive training data. If the P two-class subproblems are correctly learned by the corresponding P individual SVMs, M_i for $i = 1, \cdots, P$, then the combination of the P trained SVMs with a Max unit will produce the correct output for all the training input in $\mathcal{B}$.

Note that the module combination strategy based on the minimization and the maximization principles is called the min-max module combination in this chapter. Figure 3 illustrates the min-max module combination for a two-class problem. Here the two-class problem is first decomposed into $N_1 \times N_2$ two-class subproblems. Then each component classifier is trained on the corresponding two-class subproblem. Finally a M^3-network is constructed by combining these component classifiers. In this case, the M^3-network consists of $N_1 \times N_2$ component classifiers, N_1 Min units, and one Max unit.

3 Parallel Implementation of M^3-network

As a principle, improvement in both the feasibility and the efficiency of parallelizing in a computation process is inverse to communication among its high-level parts. The M^3-network is highly modular, that is, it has clear modules with almost no communication among them, except the final step of combining multiple results. Therefore, it is natural and efficient to run these modules simultaneously, that is, to parallelize M^3-network.

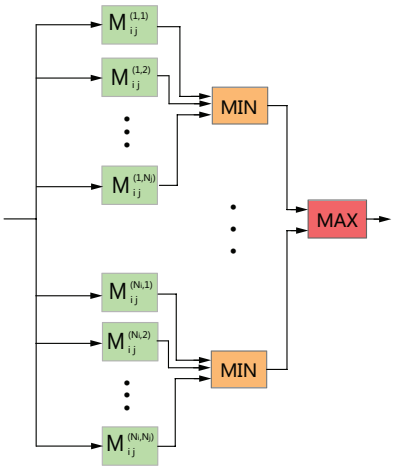


Fig. 3. Illustration of min-max module combination for a two-class problem.

3.1 Parallel Training of M^3-network

The parallel implementation of M^3-network is based on its modular structure. In the training phase, we first distribute subproblems among the computer nodes, then the nodes train component classifiers independently, and finally we collect all the trained component classifiers to finish training. In the test phase, we first distribute component classifiers, namely modules, among computer nodes, then we send the test samples to nodes and receive predictions from them, and finally we get the final predictions of the test samples through both *minimization* and *maximization* operations.

Figure 4 illustrates the parallel framework for training M^3-network. From the viewpoint of parallel programming, M^3-network falls into the category of pleasant parallel problems, and therefore master-salve framework is recommended for this category [29].

In the training phase, we need to provide each slave node with samples, which have long input vectors in large volumes. If we store all of the samples in the master node and then later send them into the slave nodes together with the subproblems, the master node will become a bottleneck for data transfer. As a solution, we distribute the samples into salve nodes that can communicate with one another through point-to-point communication. Thus, in assigning the subproblems, the master node only needs to send the slaves nodes the indexes of the samples, which greatly reduces the communication load.

Figure 5 illustrates the parallel framework of M^3-network in test phase. In the test phase, the operation of either *minimization* or *maximization* can be done with one call to *MPI_Reduce* procedure in MPI programming environment, which is well designed and works acceptably fast [29].

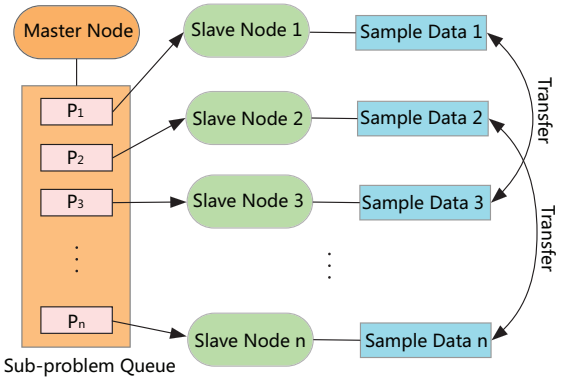


Fig. 4. The parallel framework for training M^3-network.

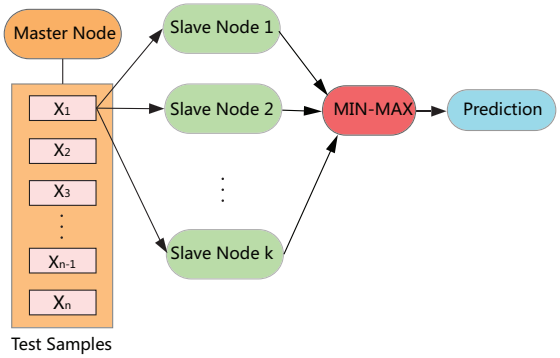


Fig. 5. The parallel framework of M^3-network in test phase.

3.2 Asymmetric and Symmetric Module Selection Strategies

The symmetric module selection strategy is an improved version of the min-max module combination method, namely asymmetric module selection strategy. This strategy works exponentially faster. Its main disadvantage is that it must work on concrete classifiers which output either 0 or 1, instead of continuous confidence values in the min-max module combination.

Let us review the min-max module combination method, and explain why we call it the asymmetric module selection strategy. Suppose the outputs of all of the modules on a test sample are positive 1 or negative 0 (see Figure 6(a)). Following the min-max module combination method, we first perform *minimization* operation along each row, then perform *maximization* operation among the rows' results. The final result is 1 in this example. Note that this result is caused by the fourth row, full of 1's as its elements. The translation of this process in feature space is that the test sample input is located within the subset $\mathcal{X}_{i,\,4}$, so that it shows positive in all the subproblems $\mathcal{T}_{ij}^{(4,\,k)}(k = 1, 2, \ldots, 5)$. The logical definition of the min-max module combination for M^3-network is:

- If there exists a row of all 1 in module's prediction matrix, the output is 1;
- Otherwise, the output is 0.

The min-max module combination method has a bias towards the output of 0, and that is why we call it asymmetric module selection.

The other combination method, the symmetric module selection strategy, is illustrated in Figure 6(b) [36]. In this method, we start from the top left element. If the element is 1, we go one step right; otherwise, we go one step down. The process continues until we go beyound the boundaries of the matrix. If we pass through the right edge of the matrix, the method outputs 1; otherwise, it outputs 0. The logical definition of this method is:

- If there exists a row of all 1 in module's prediction matrix, the output is 1;
- If there exists a column of all 0, the output is 0;
- Otherwise, the output can be either 1 or 0.

The third case is rare in solving regular pattern classification problems with M^3-network, and it doesn't need to be considered [36]. From the viewpoint of feature space, the first case means that the test sample input is located within some positive subsets, and the second case for some negative subsets, so this method make sense. Moreover, this method has no bias towards either 1 or 0, that is why we call it a symmetric module selection. Indeed, this characteristic makes it likely to perform better than the asymmetric module selection method.

The symmetric module selection strategy has a great advantage on computation complexity, compared with the asymmetric one. Suppose the training data sets for two classes are divided into m and n subsets, respectively. The complexity of the asymmetric module selection is $O(mn)$, and that of symmetric module selection is $O(m + n)$; so it is clear the symmetric module selection is one order faster than the asymmetric module selection. Further, the component classifiers

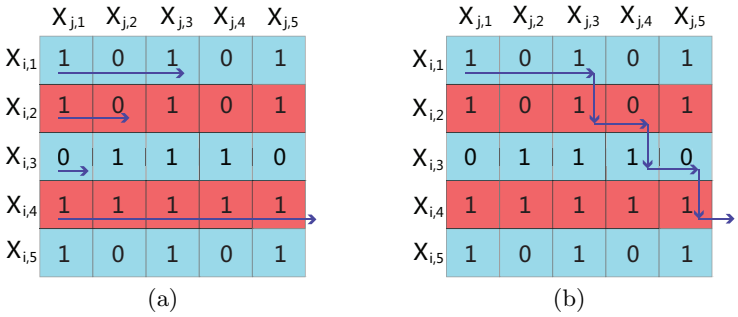


Fig. 6. Two module selection strategies: (a) asymmetric module selection and (b) symmetric module selection.

that are not visited do not need to be actually computed (see Figure 6), that is, the module combination method determines the complexity of whole test process of M^3-network. For these two reasons, the new symmetric module selection method can greatly accelerate the response speed of M^3-network, which is our main purpose.

3.3 Parallel Implementation of Symmetric Module Selection

A kind of communication exists among the modules by symmetric module selection, that is, whether a module needs to be run depends on the output of some other modules. Though we can ignore such dependency and run all the modules with the master-slave framework adopted in Section 3.1, it is not a wise choice.

The parallel pipeline framework shown in Figure 7 fits this context well. In a pipeline, some nodes do partial processing of data and then forward the partially processed results to another processing nodes down the pipeline for further processing [29].

The key issue of applying the pipeline framework is to design the task for each stage. To implement M^3-network in the style of a pipeline, we decompose the matrix of modules along the reverse diagonals; then take the modules in the same reverse diagonal as one stage, and number them from 1 to $n+m+1$, the same as the distance to the top-left cell (see the nine dotted lines in Figure 8) [33].

We build M^3-network by assigning some computer nodes to run the modules in each reverse diagonal. After that we start the system by inputting a test sample with a position status of $(1,1)$, to a computer node that is in charge of the first reverse diagonal. All the successive steps are iterative. When the test sample with the position status (r,c) arrives at the $(r+c-1)$th reverse diagonal, a computer node of that diagonal first runs the module $M_{ij}^{(r,c)}$ corresponding to the two-class subproblem $\mathcal{T}_{ij}^{(r\,c)}$. Then it updates the sample's position status with $(r, c+1)$ if the module outputs 1, or with $(r+1, c)$ otherwise. At last it passes this sample to the next reverse diagonal. The advantage of such stage

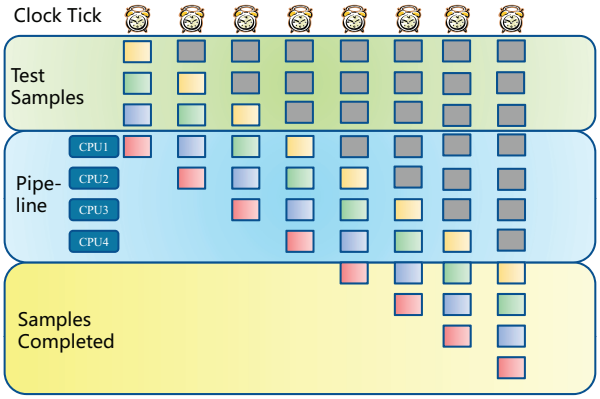


Fig. 7. Illustration of the pipeline framework

design is that each computer node can complete its job by running exactly one module, which simplifies workload balancing.

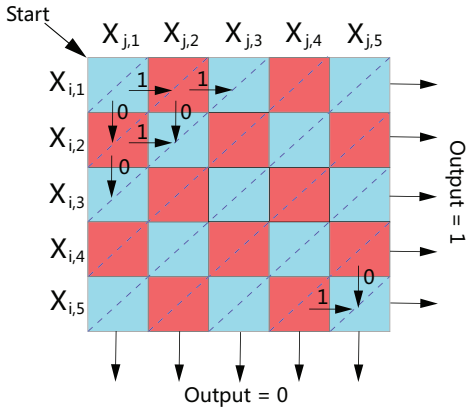


Fig. 8. Design of a pipeline for M^3-network

The number of computer nodes assigned to each reverse diagonal should agree with the number of arriving samples, for the sake of workload balance. Considering that samples leave the matrix from the right and bottom edges, from 1th to $min(n, m)$-th reverse diagonals, the number of computer nodes should be fixed; after that, the number should gradually decrease.

4 Application to Patent Documents Classification

Patents, as a social innovation system, have been an increasingly important role in science and technology improvement. As a result, automatic patent classification, as a basic data mining technique on patents, has received wide attention. A number of organizations including the European Patent Office, the Japanese Patent Office, and companies have been working on this topic [9, 12].

4.1 Patent Documents Classification

Patent documents classification takes the standard of the International Patent Classification (IPC) as a label system [12]. IPC is a complex hierarchical symbol system, where all the technological fields are divided into 8 Sections, 120 Classes, 630 Subclasses and approximately 69,000 Groups. We use M^3-network to classify Japanese patent documents on the Section level of the International Patent Classification taxonomy, which is a very large classification task [25].

We work on a Japanese patent corpus called the Japanese National Information Institutes' Testing Corpus for Information Retrieval (NTCIR), which is publicly available for research purpose[1]. The corpus consists of about 3,500,000 documents of Japanese patent applications from 1993 to 2002. A patent document is a structured text with one title and three fields, Abstract, Claim and Description (see Table 1).

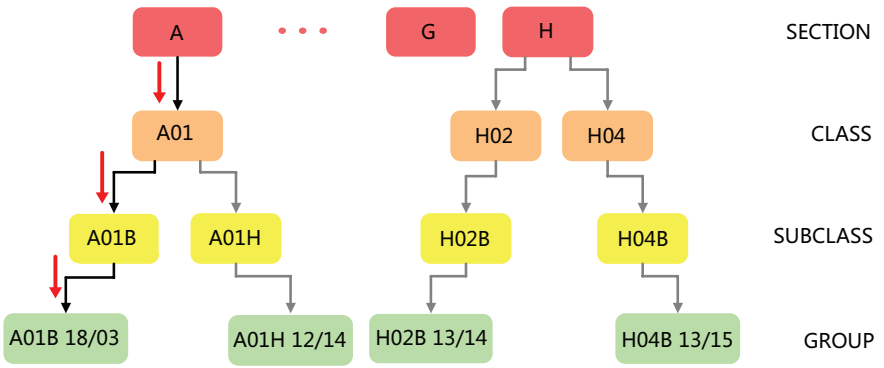


Fig. 9. Illustration of the IPC taxonomy. Here, 'A' is the Section category label, 'A01' is the Class category label, 'A01B' is the Subclass category label, and 'A01B 13/08' is the Group category label.

[1] http://research.nii.ac.jp/index-en.html

Table 1. The structure of Japanese patent documents

	PATENT-JA-UPA-1998-000001
<Bibliography>	
[publication date]	(43)【公開日】平成１０年（１９９８）１月６日
[title of invenction]	(54)【発明の名称】土壌改良方法とその作業機
<Abstract>	
[purpose]	【課題】心土破砕、特に雪上心土破砕作業の際に積雪 ...
[solution]	【解決手段】心土破砕を行うために用いるサブソイラの ...
<Claims>	
[claim1]	【請求項１】サブソイラ作業機を用いて心土破砕作業 ...
[claim2]	【請求項２】サブソイラ作業機において、そのナイフ ...
<Description>	
[technique field]	【発明の属する技術分野】本発明は、土壌改良方法とそ ...
[prior art]	【従来の技術】圃場の表面がまだ積雪に覆われている状 ...
[problem to be solved]	【発明が解決しようとする課題】心土破砕は通常春先に ...
[means of solving problems]	【課題を解決するための手段】述のような目的達成す ...
[effects of invention]	【発明の効果】以上の説明から明らかなように、本発明 ...
...	...
< Explanation of Drawing >	
[figure1]	【図１】本発明を施す圃場断面図である。
...	...

4.2 Task Decomposition with Prior Knowledge

Using hints greatly improves the effects of machine learning [1]. In this application we make use of the patents' hints, publish dates and hierarchical labels, to perform the task decomposition for M^3-network. The decomposition process, namely M^3-YC, consists of the following three steps (see Figure 10).

step 1 Divide the training data set of each Section by publish dates, each subset for one year;

step 2 Further divide the subsets by class, thus each subset for one Class published in one year;

step 3 Further randomly divide the remaining large subsets into smaller fix-sized subsets.

Note that with steps 1 and 2 removed, the above process becomes the standard M^3-network, namely M^3-Rand. In this research, our main concern is M^3-YC, and we take M^3-Rand and plain SVMs as the baseline methods.

In both M^3-Rand and M^3-YC, we can control the size of subproblems through the step of random decomposition, for example, setting the maximum number of samples in a subset. We must keep the subproblems in moderate size, neither too large for a single classifier to learn, nor so small that there are too many little subproblems. We finally decide on the maximum size of a subset to be 2000 samples, according to some pilot experiments.

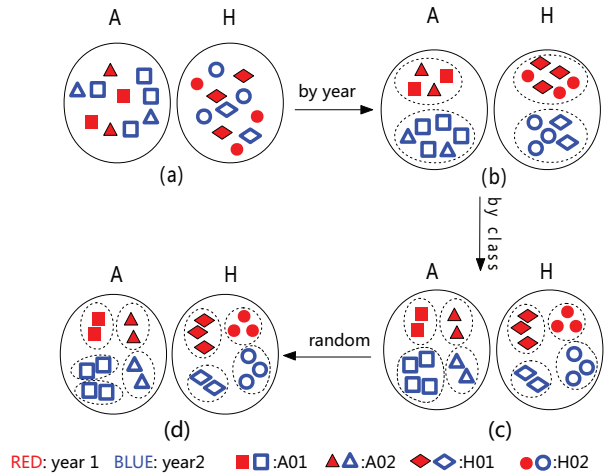


Fig. 10. Illustration of task decomposition with prior knowledge. Different attributes among the samples are shown by colors and shapes. The colors correspond to publish dates: red samples published in the same year and blue ones in another. The shapes correspond to the sub-categories: rectangles and triangles for two sub-categories of category A, and the circles and diamonds for two sub-categories of category B. The task decomposition consists of three steps, first by the prior knowledge of the publish year (b), and then by the Class category (c), and last by random for remaining large subsets (d).

4.3 Experiment Settings

Training and Test Data Sets We have an extended version of NTCIR-5 Japanese patent corpus at hand, which consists of all the patent applications published by the Japanese Patent Office from 1993 to 2002. In order to truthfully evaluate the three methods, conventional SVMs, M^3-Rand and M^3-YC, we make eight pairs of training and test data sets, creating a real-world context (see Table 2).

Table 2. Description of training and test data sets

# of years	years		set size	
for training	training	test	training	test
1	2000	2001,2002	358,072	733,570
2	1999,2000	2001,2002	714,004	733,570
3	1998–2000	2001,2002	1,055,391	733,570
4	1997–2000	2001,2002	1,386,850	733,570
5	1996–2000	2001,2002	1,727,356	733,570
6	1995–2000	2001,2002	2,064,325	733,570
7	1994–2000	2001,2002	2,415,236	733,570
8	1993–2000	2001,2002	2,762,563	733,570

Feature extraction and filtering Several steps of preprocessing need to be done before getting the vector representation of the Japanese text used by classifiers. These steps are tokenization, term filtering, and term indexing.

Tokenization generates a list of clean and informative words from raw text. We first extract the raw text from the four patent fields, Title, Abstract, Claim and Description (see Table 1), as they are most informative about the patent's content. Then we segment these texts into isolated words using the software Chasen[2]. After that, we remove the stop words (or empty words) from the results. The remaining words, namely *term*s in the research domain of text categorization, are features for the successor classification task. Table 3 shows the result we get from the example shown in Table 1.

Table 3. The terms of a Japanese patent document shown in Table 1

土壌(soil) 改良(improve) 方法(methods) 作業(working) 機(machine) % title
心土(subsoil) 破砕(crack) 雪上(snow) 心土 破砕 作業 ...% abstract
サブソイラ(subsoiler) 作業 機 心土 破砕 作業 ...%claim
発明(patent) 土壌 改良 方法 ...%description

[2] http://chasen.naist.jp/hiki/ChaSen/

Term Filtering removes the useless terms in the classification task. This shortens the length of the representation vectors, and thus cuts the computational cost and reduces generalization errors. We take χ^2_{avg} as filtering criterion [31]), and pick up 5000 top terms as features, according to our pilot experiments. Table 4 shows the top 10 terms sorted by χ^2_{avg}, most of which are technical terms representing patent documents.

Table 4. Top 10 terms selected by χ^2_{avg}

terms	explanation	χ^2_{avg}	terms	explanation	χ^2_{avg}
データ	data	10384.72	物	article	7528.72
情報	information	10199.42	含有	contain	7374.12
回路	circuit	9561.67	接続	connect	7324.43
信号	signal	8387.75	絶縁	insulation	7194.85
記録	record	7901.17	基板	baseplate	7076.72

Term Indexing generates the weights of feature terms for a sample with the real numerical vectors. We adopt the dominant methods of TFIDF [27].

$$tfidf(t,d) = n(t,d) \log \frac{|T_r|}{n_{T_r}(t)} \tag{9}$$

where t denotes a term, d denotes a document, T_r denotes the training corpus, $n(t,d)$ denotes the number of times t occurs in d, namely term frequency, and $n_{Tr}(t)$ denotes the number of documents where t occurs, namely document frequency. Table 5 shows the vector representations of selected patent documents.

Table 5. The vector representations of patent documents, where the format of the vectors adopted by SVM^{light} is taken, that is, $vetor := (dimension : value)+$.

No.	Vectors				
1	72 : 0.730	98 : 1.790	138 : 1.310	141 : 4.495	...
2	28 : 26.353	29 : 9.232	31 : 2.795	71 : 1.463	...
3	71 : 1.463	79 : 2.441	85 : 2.993	113 : 11.393	...
4	42 : 2.164	60 : 0.905	109 : 2.061	138 : 2.947	...
5	28 : 7.529	72 : 6.577	139 : 8.103	167 : 8.728	...

Computational Platform A Lenovo cluster system consisting of three fat nodes and thirty thin nodes run all of our experiments. Each fat node has 32 GB of RAM and two 3.2-GHz quad-core CPUs, while each thin node has 8 GB of RAM and two 2.0-GHz quad-core CPUs. Experiments with the conventional SVMs were performed on the fat nodes because they need large memory, while experiments with the M^3-network were done on the thin nodes because each subproblem was small and a lot of processors were required for parallel training.

Training Component Classifier M^3-network is a general framework for pattern classification. The user can select suitable component classifiers according to the classification task to be solved and available computing resource. There are several alternative choices, such as multilayer neural networks[21], k-nearest-neighbor algorithm[35, 30], and SVMs [23]. Here we select conventional SVMs with linear kernel and use SVM^{light}, an implement of SVM by Joachims [15]. SVM^{light} actually plays two roles in this chapter, the baseline method and the component classifiers for M^3-network.

Performance Measurement The conventional measurement of accuracy in machine learning domain makes less sense for most text categorization tasks; instead, we ususally use the function F_1 [27]. The formular F_1 of one class is :

$$F_1 = \frac{2PR}{P+R} \tag{10}$$

$$P = \frac{TP}{TP+FP} \tag{11}$$

$$R = \frac{TP}{TP+FN} \tag{12}$$

where TP is the number of classifier's True Positive predictions, FP for False Positive, and FN for False Negative. P is named the precision, and R is named the recall. In practice, there are the two following versions of F_1, depending on the methods of integrating individual results:

$$micro - F_1 = \frac{2PR}{P+R} \tag{13}$$

$$macro - F_1 = \underset{c \in C}{avg} \frac{2P_cR_c}{P_c+R_c} \tag{14}$$

where P and R are computed with all the classes, while P_c and R_c are computed on only class c. For example, sample s actually belongs to classes c_1 and c_2, while the classifier's predictions is c_2 and c_3. Then in computing $micro - F_1$, TP, FP and FN will be increased by 1, while in computing P_{c1} and R_{c1} for $macro - F_1$, only FN will be increased by 1.

4.4 Results and Discussions

Accuracy Figure 11 is the experimental results, with micro-F_1 and macro-F_1 used as accuracy measurements. The following conclusions can be drawn from our results:

(a) On the aspect of test accuracy, the two M^3-SVM methods, M^3-Rand and M^3-YC, are both superior to conventional SVMs. We can learn from the training scores that conventional SVMs with linear kernel are unable to learn the training set completely, because its micro-F_1 and macro-F_1 are

only about 80%. On the contrary, as an ensemble learning algorithm, M^3-SVMs can generate a powerful classifier by combining simple classifiers. As a result, M^3-SVMs have fulfilled the learning on all the training sets with accuracies of nearly 100%.

(b) M^3-Rand and M^3-YC show superior robustness to conventional SVMs over dated samples. Along the moving backward of the starting time point of training set, more and more dated samples are added, thus the performance of conventional SVMs decreases. Contrarily, the performance of two M^3-SVMs increases at the same time. Though their performance deceases a little in the 5th year point and 7th year point. This does not affect the curves' overall trend.

(c) M^3-YC over-performs M^3-rand on each year point, which indicates that incorporating prior knowledge of the publishing date and the Class category into task decomposition will undoubtedly improve classification performance.

Time Cost As mentioned before, M^3-SVMs (both M^3-Rand and M^3-YC) have the merit of parallel computing, which greatly speeds up learning. However, we are sharing the Lenovo parallel computer system with other users while performing the experiments, so the accurate time cost of M^3-SVMs could not be measured. The time cost of conventional SVMs has been recorded as it didn't involve parallel running.

We evaluate the time cost of M^3-YC using the following formula:

$$t_{M^3} = \frac{n_{mod} \times t_0}{n_{cpu}} \tag{15}$$

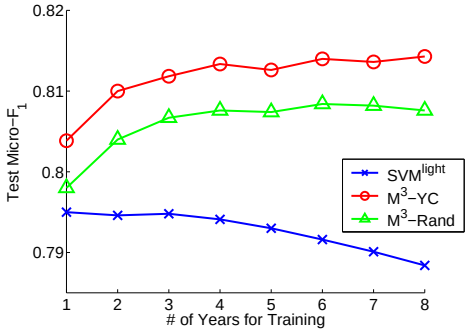
where n_{mod} is the number of modules (or subproblems), t_0 is the average time cost of per module, and n_{cpu} is the number of CPUs. During our experiments, n_{mod} have been recorded automatically and n_{cpu} is 30 on the Lenovo cluster system. As for t_0, it must be measured by experiment, and the eventual value we get is 0.025 seconds per module.

Figure 12 is the time costs of conventional SVMs and M^3-YC in our experiments. From this figure, we can know that the M^3-YC's time cost is only about 1/10 of the SVM's, which agrees with our expectation well.

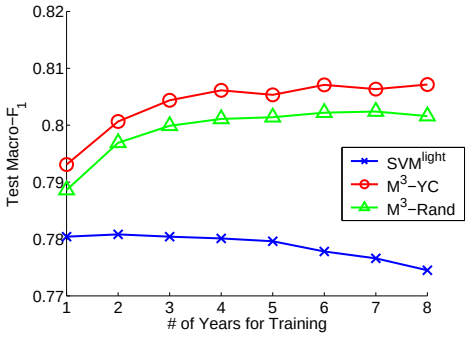
5 Conclusions

In this chapter, we presente a parallel and modular pattern classification framework for large-scale problems. The framework works in a modular manner, and has several advantages in solving large-scale problems. On the one hand, massively parallel and distributed training is easily implementable because of its modularity. On the other hand, it has a balanced performance on all classes because of various task decomposition strategies.

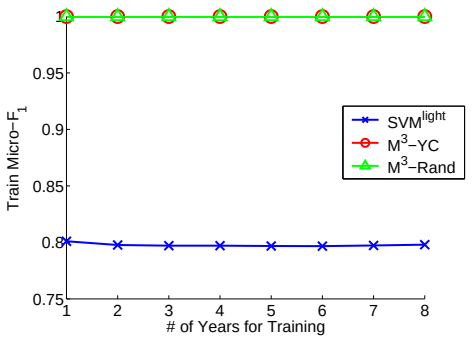
The experiments are conducted on a large-scale Japanese patent corpus. Taking into account prior/domain knowledge, we partition large training data sets



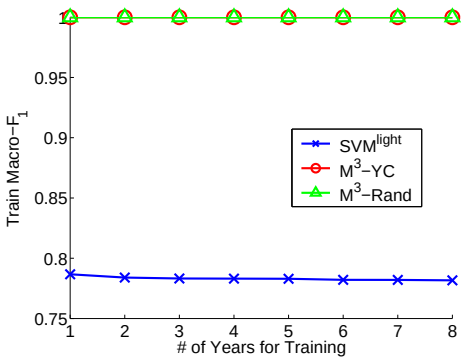
(a)



(b)



(c)



(d)

Fig. 11. Performance comparison of conventional SVMs, M^3-Rand and M^3-YC: (a) micro-F_1 on test data; (b) macro-F_1 on test data; (c) micro-F_1 on training data; and (d) macro-F_1 on training data.

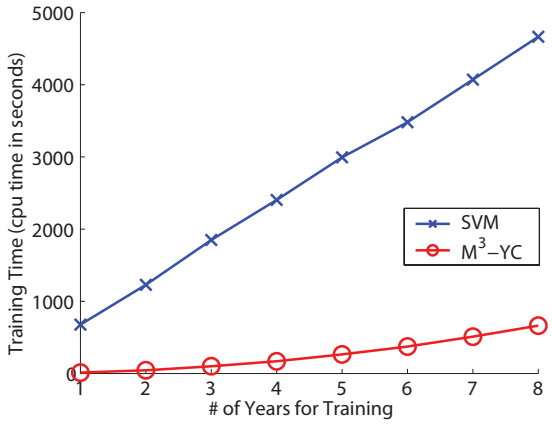


Fig. 12. Training time comparison of conventional SVMs and M^3-YC.

into smaller subsets according to the publishing date and the subclass. The experimental results show the effectiveness of the proposed framework, and demonstrate that M^3-SVMs is superior to conventional support vector machines in solving such complex problems.

As more and more large-scale applications appear in the fields of machine learning and pattern recognition, there is greater need for parallel and distributed methods. M^3-network uses simple task decomposition methods and efficient module selection strategies and can be implemented easily in practice. We recognize that incorporating prior/domain knowledge into task decomposition is a reliable way to improve the learning efficiency and the generalization performance of M^3-network. We believe that the proposed parallel and modular framework will be very useful for solving complex classification problems with very large data sets.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 60773090 and Grant No. 90820018), the National Basic Research Program of China (Grant No. 2009CB320901), and the National High-Tech Research Program of China (Grant No. 2008AA02Z315).

References

1. Abu-Mostafa, Yaser S.: Machines that learn from hints, Scientific American, vol. 272, no. 4, pp. 64–69, 1995.
2. Anand, R.; Mehrotra, K.G.; Mohan, C.K.: An improved algorithm for neural-network classification of imbalanced training sets, IEEE Transcations on Neural Networks, vol 4, no. 6, pp. 962-969, 1993.

3. Angiulli, F; Folino, G.: Distributed nearest neighbor-based condensation of very large data sets, IEEE Transactions on Knowledge and Data Engineering, vol. 19, pp. 1593-1606, 2007.
4. Cao, L. J., Keerthi, S. S., Ong, C. J., Zhang, J. Q., Periyathamby, U., Fu, X. J. and Lee, H. P.: Parallel sequential minimal optimization for the training of support vector machines, IEEE Trans Neural Network, pp. 1039-1049, 2006.
5. Chen, K., Lu, B. L. and Kwok, J. T.: Efficient classification of multi-label and imbalanced data using min-max modular classifiers, Proc. of IEEE/INNS International Joint Conference on Neural Networks, pp. 1770-1775, 2006.
6. Chu, X.L., Ma, C., Li, J., Lu, B.L., Utiyama, M., and Isahara, H.: Large-scale patent classification with min-max modular support vector machines, Proc. of IEEE/INNS International Joint Conference on Neural Networks, pp. 3973-3980, 2008.
7. Collobert, Ronan; Bengio, Samy; Bengio, Yoshua.: A parallel mixture of SVMs for very large scale problems, Neural Computation, vol. 14, Issue 5, pp. 1105-1114, 2002.
8. Dong, J., Krzyzak, A. and Suen, C. Y.: Fast SVM training algorithm with decomposition on very large data sets, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 4, pp. 603-618, 2005.
9. Fall, C.J., T orcsvári, A., Benzineb, K., Karetka, G.: Automated categorization in the international patent classification, ACM SIGIR Forum ACM Press New York, NY, USA, vol. 37, pp. 10-25, 2003.
10. Fan, Z.G. and Lu, B.L.: Multi-view face recognition with min-max modular SVMs, Lecture Notes in Computer Science, Springer, vol. 3611, pp. 396-401, 2005.
11. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., and Lin, C.J.: LIBLINEAR: A Library for large linear classification. Journal of Machine Learning Research, vol. 9, pp. 1871-1874, 2008.
12. Fujii, A., Iwayama, M., Kando, N. : Introduction to the special issue on patent processing, Information Processing and Management, pp. 1149–1153, 2007.
13. Hazan, T., Man, A., and Shashua, A. : A parallel decomposition solver for SVM: distributed dual ascend using Fenchel duality, Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8, 2008.
14. Ferreira, L.V.; Kaszkurewicz, E.; Bhaya, A. : Parallel implementation of gradient-based neural networks for SVM training, Proc. of International Joint Conference on Neural Networks, pp. 339–246, 2006.
15. Joachims, T.: Making large-scale support vector machine learning practical. Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, pp. 169-184, 1998.
16. Keerthi, S. S. and DeCoste, D. : A modified finite Newton method for fast solution of large scale linear SVMs, Journal of Machine Learning Research, vol. 6, pp. 341-361, 2005.
17. Kugler, M; Kuroyanagi, S., and Nugroho, A.S.: CombNET-III with nonlinear gating network and its application in large-scale classification problems, IEICE Transactions on Information and Systems, vol. E91D, no. 2, pp. 286-295, 2008.
18. Larkey, L.: Some issues in the automatic classification of US patents, Proc. of AAAI-98 Workshop on Learning for Text Categorization, Technical Report WS-98-05, pp. 87-90, 1998.
19. Liu, F. Y., Wu, K., Zhao, H. and Lu, B. L.: Fast text categorization with a min-max modular support vector machine, Proc. IEEE/INNS International Joint Conference on Neural Networks, pp. 570–575, 2005.
20. Lu, B. L. and Ito, M.: Task decomposition based on class relations: a modular neural network architecture for pattern classification, Biological and Artificial Com-

putation: From Neuroscience to Technology, Lecture Notes in Computer Science, Springer, vol. 1240, pp. 330-339, 1997.
21. Lu, B. L. and Ito, M.: Task decomposition and module combination based on class relations: A modular neural network for pattern classification, IEEE Transactions on Neural Networks, Vol. 10, pp. 1244-1256, 1999.
22. Lu, B. L., Ma, Q., Ichikawa, M. and Isahara, H.: Efficient part-of-speech tagging with a min-max modular neural-network model, Applied Intelligence, vol. 19, no. 1, pp. 65-81, 2003.
23. Lu, B. L., Wang, K. A., Utiyama, M., and Isahara, H.: A part-versus-part method for massively parallel training of support vector machines, Proc. of IEEE/INNS International Joint Conference on Neural Networks, pp. 735-740, 2004.
24. Lu, B. L and J. Li: A Min-Max modular network with gaussian-zero-crossing function, Studies in Computational Intelligence, vol. 35, pp. 285-313, 2008.
25. Lu, B. L, Wang, X. L. and Utiyama, M.: Incorporating prior knowledge into learning by dividing training data, Journal of Frontiers of Computer Science in China, pp. 109-122, 2009.
26. Lu, Y. and Roychowdhury, V.: Parallel randomized sampling for support vector machine and support vector regression, Knowledge and Information Systems, vol. 14, pp. 233-247, 2008.
27. Sebastiani, F.: Machine learning in automated text categorization, ACM Computing Surveys, vol. 34, pp. 1-47, 2002.
28. Sun, Y. M., Kamel, M. S. and Wong, A. K. C: Cost-sensitive boosting for classification of imbalanced data, Pattern Recognition, vol. 40, pp. 3358-3378, 2007.
29. Wilkinson, B. and Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstation and Parallel Computers, Prentice Hall Inc., 1999.
30. Wu, K., Lu, B. L., Uchiyama, M., and Isahara, H.: An empirical comparison of min-max-modular k-NN with different voting methods to large-scale text categorization, Soft Computing, vol. 12, no. 7, pp. 647-655, 2008.
31. Yang, Y.M. and Pedersen, J. O.: A comparattive study on feature selection in text categorization, Proc. of International Conference on Machine Learning, pp. 187-196, 1997.
32. Yang, Y. and Lu, B. L.: Prediction of protein subcellular multi-locations with a min-max modular support vector machine, Lecture Notes in Computer Science, Springer, pp. 646-651, 2005.
33. Ye, Z. F., Lu, B. L. and Hui, C.: Patent classification using parallel Min-Max modular support vector machine, Autonomous Systems, Springer, pp. 157-167, 2008.
34. Zanni, L; Serafini, T; Zanghirati, G.: Parallel software for training large scale support vector machines on multiprocessor systems, Journal of Machine Learning Research, vol. 7, pp. 1467-1492, 2006.
35. Zhao, H. and Lu, B. L.: A modular k-nearest neighbor classification method for massively parallel text categorization, Lecture Notes in Computer Science, Springer, vol. 3314, pp. 867–872, 2004.
36. Zhao, H. and Lu, B. L.: Improvement on response performance of Min-Max modular classifier by symmetric module selection, Lecture Notes in Computer Science, Springer, vol. 3497, pp: 39-44, 2005.