

A Parallel Learning Algorithm for Text Categorization on PIRUN Beowulf Cluster

Canasai Kruengkrai and Chuleerat Jaruskulchai

Intelligent Information Retrieval and Database Laboratory,
Department of Computer Science, Faculty of Science,
Kasetsart University, Bangkok, Thailand
{g4364115,fscichj}@ku.ac.th

Abstract. Text categorization is the process of classifying documents into predefined categories or classes based on their content. Since text data rapidly increase on the Internet, the scalability of the algorithm is required to handle such massive data. In this paper, we propose a parallel learning algorithm for text categorization based on the combination of the Expectation-Maximization (EM) algorithm and the naive Bayes. Our experiment performed on a 72 nodes Beowulf cluster called PIRUN. The preliminary experimental results show that our parallel implementation has reasonable speedup characteristics.

1 Introduction

Text categorization has become one of the most important techniques in text mining. The task is to automatically classify documents into predefined categories or classes based on their content. Many algorithms have been developed to deal with automatic text categorization task. One of the common methods is the naive Bayes. Although the naive Bayes works well in many studies [7][9][10], it requires a large number of labeled training documents for learning accurately. In the real world task, it is very hard to obtain the large labeled documents, which are mostly produced by humans. To overcome this shortage, Nigam et al. [13] apply the Expectation-Maximization (EM) algorithm to the text categorization problem. The EM algorithm uses both labeled and unlabeled documents for learning. Nevertheless, the EM algorithm is too slow when it performs on very large document collections. Furthermore, the text data rapidly increase on the Internet. The scalability of the algorithm is required to handle such massive data.

The parallel processing is an interesting technique for scaling up learning algorithms. For example, the parallel algorithms are applied for mining association rules [1] and classification based on decision tree [8]. Several researches study techniques for parallelizing clustering algorithms. Ruocco and Frieder [15] propose parallel single-link and single-pass algorithm for clustering documents worked on an Intel Paragon. Dhillon and Modha [3] introduce an effective parallelization of the k-means clustering algorithm implemented on an IBM

POWERparallel SP2. Forman and Zhang [4] also present a general technique for parallelizing a class of center-based clustering algorithms including k-means, k-harmonic means, and EM algorithm performed on existing network structures (LAN). The similar ideas of the last two works are to use data parallelism and limit the communication among processes to only some parameters.

In this paper, we propose a new parallel learning algorithm for the text categorization. In particular, the parallel algorithm is based on the combination of the EM algorithm and the naive Bayes using message-passing model. Our experiment performed on a 72 nodes BeoWulf cluster called PIRUN (Pile of Inexpensive and Redundant Universal Nodes) Cluster at Kasetsart University. The preliminary experimental results show that our parallel implementation has reasonable speedup characteristics.

The paper is organized as follows. In Section 2, we describe how text data is represented by using the vector space model. In Section 3, we briefly review the probabilistic framework including the naive Bayes and the EM algorithm. In Section 4, we present the parallel implementation of the EM algorithm and its complexity analysis. In Section 5, we present the experimental results. Finally, Section 6 is the conclusion of our work.

2 Text Document Representation

Typically, text documents are unstructured data. Before learning process, we must transform them into a representation that is suitable for computing. We use the vector space model [2], most widely used in Information Retrieval, to represent the text documents. This model is equivalent to an attribute value representation used in Machine Learning [12]. Each distinct word is a feature (or index term) and the number of times the word occurs in the document is its value (or term frequency).

Let us describe how to construct the vector space model from a document collection. First, we parse the document collection to extract unique words and prune non-content words (or stop-words), low- and high-frequency words. Then, we obtain $w = (w_1, w_2, \dots, w_k, \dots, w_V)$, where V is the number of the unique words within the collection. In the vector space model, we ignore the sequence in which the word occurs. Next, each document is represented by a vector $d_i = (w_{i1}, w_{i2}, \dots, w_{iV})$, where w_{ik} is the frequency of k th word in a document d_i . Finally, we map the entire document vectors to a matrix called the document-word matrix (see Figure 1). Each row of the matrix corresponds to a document vector. The columns of the matrix correspond to the unique terms in the document collection.

3 Probabilistic Framework for Text Categorization

Suppose that we have training documents $D = d_1, \dots, d_N$, where each document is labeled by $y_i \in c_1, \dots, c_M$. We write $y_i = c_j$ when a document d_i belongs to a class c_j . We use Θ to denote the parameters of the model. We assume

	w_1	w_2	\dots	w_k	\dots	w_V
d_1	w_{11}	w_{12}	\dots	w_{1k}	\dots	w_{1V}
d_2	w_{21}	w_{22}	\dots	w_{2k}	\dots	w_{2V}
\dots	\dots	\dots	\dots	\dots	\dots	\dots
d_k	w_{i1}	w_{i2}	\dots	w_{ik}	\dots	w_{iV}
\dots	\dots	\dots	\dots	\dots	\dots	\dots
d_N	w_{N1}	w_{N2}	\dots	w_{Nk}	\dots	w_{NV}

Fig. 1. A document-word matrix used for representing a document collection

that documents are generated by a mixture of multinomial model, and there is a one-to-one correspondence between mixture components and classes. In this model, each document d_i is generated by choosing a mixture component with the class prior probabilities $P(c_j|\Theta)$, and having this mixture component generate a document according to its own parameters, with distribution $P(d_i|c_j; \Theta)$. Thus, we can write:

$$P(d_i|\Theta) = \sum_{j=1}^M P(c_j|\Theta)P(d_i|c_j; \Theta) . \quad (1)$$

3.1 Naive Bayes

The naive Bayes uses the maximum a posteriori (MAP) estimate for learning a classifier. It assumes that the occurrence of each word in a document is conditionally independent of all other words in that document given its class. Using the assumption, the probability of a document given its class becomes:

$$P(d_i|c_j; \Theta) = P(w_{i1}, \dots, w_{i|d_i}|c_j; \Theta) \propto \prod_{k=1}^{|d_i|} P(w_{ik}|c_j; \Theta) . \quad (2)$$

Therefore, the parameters of the model are the conditional probabilities $\theta_{w_k|c_j} = P(w_k|c_j; \Theta)$, and the prior probabilities $\theta_{c_j} = P(c_j; \Theta)$. That is:

$$\Theta = \{\theta_{w_1|c_1}, \dots, \theta_{w_V|c_M}; \theta_{c_1}, \dots, \theta_{c_M}\} . \quad (3)$$

Let $\hat{\Theta}$ be the estimates of Θ . The parameters $\theta_{w_k|c_j}$ can be estimated by using *Laplace smoothing* that adds one to all the word counts to avoid probabilities of zero. Thus we obtain:

$$\hat{\theta}_{w_k|c_j} = \frac{1 + \phi(w_k, c_j)}{V + \sum_{w' \in w} \phi(w', c_j)} , \quad (4)$$

where $\phi(w_k, c_j)$ is the number of times that a word w_k occurs in the training documents for a class c_j . The parameters θ_{c_j} can be estimated as follows:

$$\hat{\theta}_{c_j} = \frac{\phi(d_i, c_j)}{N} , \quad (5)$$

where $\phi(d_i, c_j)$ is the number of training documents d_i that are assigned to a class c_j . Given estimates of these parameters calculated from the training documents, we can classify an unseen document to a single class that the posteriori probability is highest; $\operatorname{argmax}_j P(c_j|d_i; \hat{\Theta})$. It can be calculated by applying Bayes' formula:

$$\begin{aligned}
P(c_j|d_i; \hat{\Theta}) &\propto P(c_j|\hat{\Theta})P(d_i|c_j; \hat{\Theta}) \\
&= P(c_j|\hat{\Theta}) \prod_{k=1}^{|d_i|} P(w_{ik}|c_j; \hat{\Theta}) \\
&= P(c_j|\hat{\Theta}) \prod_{k=1}^V P(w_k|c_j; \hat{\Theta})^{\phi(w_k, d_i)} , \tag{6}
\end{aligned}$$

where $\phi(w_k, d_i)$ is the number of times that a word w_k occurs in a document d_i .

3.2 Expectation-Maximization

One drawback of the naive Bayes is that it requires a large set of the labeled training documents for learning accurately. The cost of labeling documents is expensive, while unlabeled documents are commonly available. By applying the EM algorithm, we can use the unlabeled documents to augment the available labeled documents.

The EM algorithm is a general technique for maximum likelihood or maximum a posteriori estimation in incomplete-data problems [11]. In our task, the class labels of the unlabeled documents are considered as the missing values. The document collection D now consists of the disjoint subsets of the labeled and the unlabeled documents: $D_l \cup D_u$. The probability function of all the documents becomes:

$$\begin{aligned}
P(D|\Theta) &= \prod_{d_i \in D_l} P(y_i = c_j|\Theta)P(d_i|y_i = c_j; \Theta) \\
&\times \prod_{d_i \in D_u} \sum_{j=1}^M P(c_j|\Theta)P(d_i|c_j; \Theta) . \tag{7}
\end{aligned}$$

For the labeled documents, the generating component is given by labels y_i , we do not need to refer to all mixture components [13]. As described in Section 3.1, we use the MAP estimate, $\operatorname{argmax}_{\Theta} P(\Theta|D)$, for learning a classifier. By making use of the Bayes' formula and Equation 7, we obtain the MAP estimate of Θ , which is equivalent to the value of Θ that maximizes the log-posteriori:

$$\begin{aligned} \log P(\Theta|D) &= \sum_{d_i \in D_l} \log(P(y_i = c_j|\Theta)P(d_i|y_i = c_j; \Theta)) \\ &+ \sum_{d_i \in D_u} \log\left(\sum_{j=1}^M P(c_j|\Theta)P(d_i|c_j; \Theta)\right) + \log(P(\Theta)) . \end{aligned} \quad (8)$$

Equation 8 cannot be computed directly, because the second term contains a log of summations. Here we introduce the class indicator variables Z , where each $z_{ij} \in Z$ is defined to be one or zero according as d_i does or does not come from the j th class. By using the class indicator variables Z , we can write the complete-data log-posteriori in the form:

$$\log P_c(\Theta|D; Z) = \sum_{d_i \in D} \sum_{j=1}^M z_{ij} \log(P(c_j|\Theta)P(d_i|c_j; \Theta)) + \log(P(\Theta)) , \quad (9)$$

where the log of the priori $P(\Theta)$ is approximated by using Dirichlet distribution [13]. Since we do not know the exact values of Z , we instead work with their expectation. The algorithm finds a local maximum of the complete-data log-posteriori by iterating the following two steps:

$$\begin{aligned} \text{E-step} : \hat{Z}_{(k+1)} &= E[Z|D; \hat{\Theta}_{(k)}] \\ \text{M-step} : \hat{\Theta}_{(k+1)} &= \operatorname{argmax}_{\Theta} P(\Theta|D; \hat{Z}_{(k+1)}) , \end{aligned} \quad (10)$$

where the E-step is the current parameter estimates of probabilistic labels for the unlabeled documents calculated by Equation 6, and the M-step is the new MAP estimates for the parameters calculated by Equation 4 and 5.

4 Parallel Implementation

In this section, we describe the parallel implementation of the EM algorithm for text categorization. We employ the Single Program Multiple Data (SPMD) model using message-passing in our parallelization. We assume that we have P processors, where each processor is assigned a unique rank between 0 and $P - 1$ and has an individual local memory. The processors communicate with each other by using MPI (Message-Passing Interface) library.

The EM algorithm starts by using the naive Bayes to initialize the parameters. The E- and M-step are iterated until the change of the $\log P_c(\Theta|D; Z)$ is less than a predefined threshold. The E-step almost dominates the execution time on each iteration, since it estimates the class labels for all the unlabeled documents. Fortunately, we observe that this step is inherently data parallel, because if the parameters Θ are available for each processor, the same operation can be performed on different documents simultaneously. We parallelize

Input: Documents both labeled D_l and unlabeled D_u

1. Processor P_0 builds the initial global parameters Θ_g from only the labeled documents D_l , and broadcasts them to all processors
2. Processor P_r reads the unlabeled documents D_u based on its responsibility from a disk
3. Iterate until convergence
 - (a) E-step: Each processor P_r estimates the class of each document by using the current global parameters Θ_g
 - (b) M-step: Each processor P_r re-estimates its own local parameters Θ_l given the estimated class of each document
 - (c) Sum up the local parameters Θ_l to obtain the new global parameters Θ_g and return them to all processors

Output: The global parameters Θ_g of the model

Fig. 2. The outline of the parallel EM algorithm for text categorization

the loop by evenly distributing the documents across processors. If we partition the N documents into P blocks, each processor handles roughly N/P documents. In other words, P_r is given a responsibility for documents D_i , where $i = (r)(N/P) + 1, \dots, (r + 1)(N/P)$.

Let Θ_l and Θ_g be the local and the global parameters of the model, respectively. In the training process, the processor P_0 computes the global parameters Θ_g from only the labeled training documents. Then, the processor P_0 distributes them to the available processors by using `MPI_Bcast`. We assign this task to only processor P_0 , because the naive Bayes can learn in constant time. Next, each processor P_r uses the current global parameters Θ_g to label the unlabeled documents for its partition. Finally, each processor P_r calculates its local parameters Θ_l and calls `MPI_Allreduce` to sum up the local parameters Θ_l to obtain the new global parameters Θ_g . The algorithm uses these parameters as the current parameters in the next iteration. Since each processor has the same global parameters Θ_g , it can independently decide when it should exit the loop. Figure 2 gives the outline of the parallel EM algorithm.

The `MPI_Allreduce` function is a global communication operation that the result of the reduction operation is available in all processes [6]. The parameters of our model in Equation 3 are the probability estimates consisting of word and document counts among different classes. As a result, the local parameters Θ_l achieve the global parameters Θ_g by simply using `MPI_Allreduce` with the reduction operation `MPI_SUM`. Our algorithm design can avoid the network bottleneck, because there are only the parameters that exchange across processes. In the test process, we use the final global parameters Θ_g to classify the test documents that are evenly partitioned for each processor as in the E-step.

Complexity Analysis. The time complexity of the sequential EM algorithm can be calculated as follows. In training process, the initial step requires $V N_{train}$

for estimating the parameters from the labeled documents. The E-step takes VMN_{un} , since the class estimation is performed on the unlabeled documents. The M-step takes VN_{un} . Let I be the number of iterations. As described above, the training process is dominated by the loop. The training process requires $VN_{train} + I(VMN_{un} + VN_{un})$, which can be approximated by $O(IVMN_{un})$ for the large number of N_{un} . In test process, it requires $O(VMN_{test})$ similar to the E-step. Therefore, we obtain the overall time complexity $O(IVMN_{un} + VMN_{test})$. The space complexity of the algorithm requires $2(VM + M)$ for storing the current and the updated parameters, and VN_s for storing the subsets of the document collection on demand. We finally obtain the total space complexity $O(2(VM + M) + VN_s)$.

For parallel processing, since each processor handles only N/P documents, the computational time decreases to at most $O(IVM(N_{un}/P) + VM(N_{test}/P))$. The communication time for exchanging the parameters is $O(I(VM + M)T_{data})$, where T_{data} is the transmission time for the parameters. Consequently, the overall parallel time complexity is estimated as:

$$O(IVM(N_{un}/P) + VM(N_{test}/P) + I(VM + M)T_{data}) , \quad (11)$$

and the space complexity reduces to $O(2(VM + M) + V(N_s/P))$ for each node.

5 Experimental Results

In this section, we provide empirical evidence that our parallel algorithm design can improve efficiency. We implemented our algorithm on PIRUN Beowulf Cluster at Kasetsart University. PIRUN Cluster consists of 72 nodes connected with Fast Ethernet Switch 3COM SuperStack II. Each node is a 500 MHz Pentium III with 128 Mbytes of RAM, has no local hard disk, and use Linux operating system (see <http://pirun.ku.ac.th> for details).

5.1 Data Collection and Performance Measures

The 20 Newsgroups collection [7][10][13] was used in our experiments. It consists of 20000 articles divided evenly among 20 different UseNet discussion groups. After removing stop-words, low- and high-frequency words, we obtained 11350 unique words. We split 30% of the collection to form the labeled training set (6000 documents), 50% to form the unlabeled training set (10000 documents), and the remaining 20% to form the test set (4000 documents). Each document set is represented by a document-term matrix. In practice, the document-word matrix is very sparse, having mostly zero entries. To minimize the overhead of loading matrix files from a hard disk via the network, we compressed each document-word matrix by using the Scalar ITPACK compression method [5], which also capable helps the program to reduce memory consumption.

We measured the elapsed time (disk accesses included) from start to complete the task. To evaluate performance, we examined the speedup (S) of our

algorithm design. The speedup is the ratio of the execution time for learning and classifying a document collection on a single processor to execution time for the same tasks on P processors. Thus, the speedup of the parallel EM algorithm can be approximated as follows:

$$S = \frac{O(IVMN_{un} + VMN_{test})}{O(IVM(N_{un}/P) + VM(N_{test}/P) + I(VM + M)T_{data})}, \quad (12)$$

which increases linearly with P , if we have the large numbers of N_{un} and N_{test} .

5.2 Results

Figure 3 shows response times of the algorithm on different numbers of processors ranging from 1 to 16 processors. Figure 4 demonstrates the relative speedups. We also varied the unlabeled training documents into three subsets to observe the effects of problem sizes on the performance. The algorithm performs 6, 7, 8 iterations to satisfy the convergence criterion on each subset. From our experiments, the time of the initial step using the naive Bayes does not affect the performance. It takes less than 18 seconds even though it uses the entire labeled training documents. The computational time of the algorithm is mostly dominated by the loop in the training process. As we analyzed above, the speedup increases linearly in some cases. For the largest unlabeled set, it achieves the relative speedups of 3.72, 7.16, and 12.16 on 4, 8, and 16 processors, respectively. The other unlabeled document sizes give the same trend.

We observed that the speedup significantly drops from the linear curve when the number of processors ranges from 8 to 16, because of increased overheads.

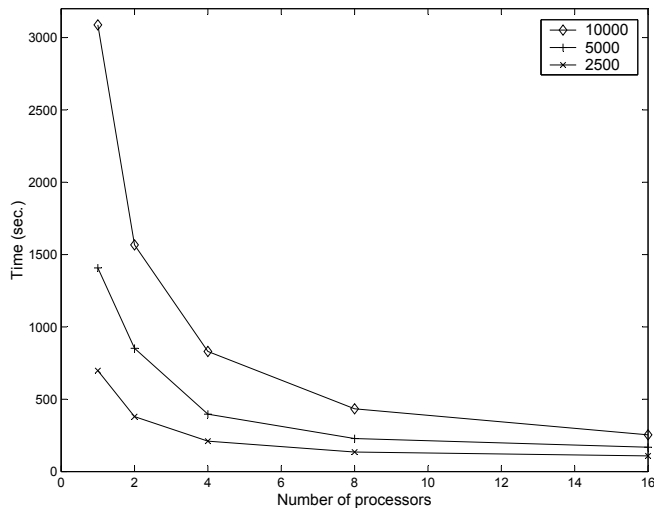


Fig. 3. The execution time of the parallel EM algorithm with 2500, 5000, and 10000 unlabeled documents

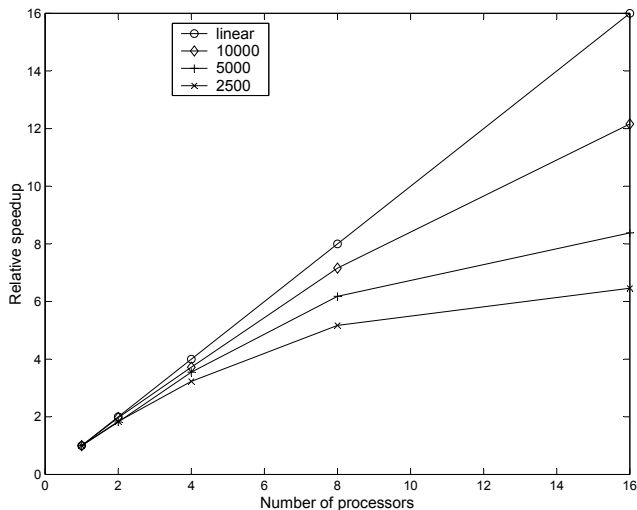


Fig. 4. The relative speedups of the parallel EM algorithm corresponding to Figure 3. When the problem sizes are scaled up, the speedup improves from 5.17 to 7.16 on 8 processors, and from 6.46 to 12.16 on 16 processors

This is a normal situation when the problem size is fixed as the number of processors increases. However, it can be solved by scaling the problem size. In our experiment, for example, the speedup improves from 5.17 to 7.16 on 8 processors for scaling the unlabeled document sizes from 2500 to 10000 documents, and from 6.46 to 12.16 on 16 processors. It can be seen that our parallel algorithm yields better performance for the larger data sets.

In order to ensure that the EM algorithm with the unlabeled documents can improve classification accuracy, we also re-examined its effectiveness using 10000 unlabeled documents (see [13][14] for the comprehensive study on the effectiveness). For each parallel execution, the classification result is equivalent to its sequential execution. In our experiment, we observed that the EM algorithm significantly outperforms the naive Bayes when the amount of the labeled documents is small. The results are shown in Figure 5. For example, the EM algorithm achieves 31% accuracy while the naive Bayes reaches 19% accuracy at 20 labeled documents (or one document per category). With 100 labeled documents, the EM algorithm achieves 57% accuracy while the naive Bayes reaches 32% accuracy. The two approaches begin to converge when the amount of the labeled documents is large. There is a concern of the straightforward use of the EM algorithm for text classification task. The EM algorithm with unlabeled documents can hurt accuracy on some data collections, because the generative model is not representative. However, this problem can be solved by using more complex generative model [14].

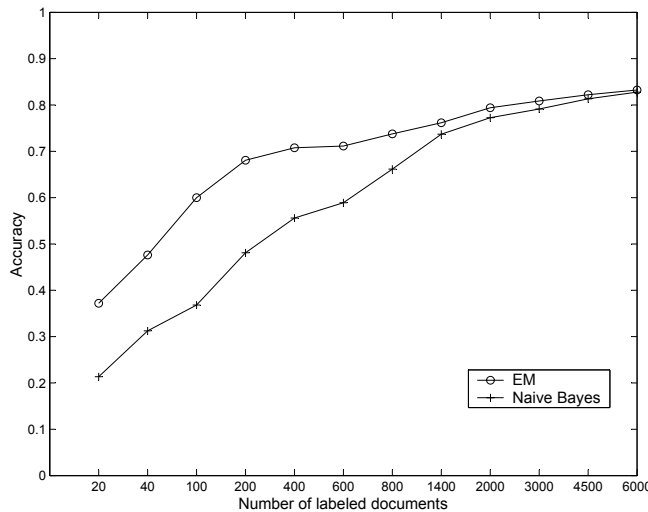


Fig. 5. Classification accuracy vs. the number of labeled training documents on 20 News- groups. The EM algorithm accesses 10000 unlabeled documents in learning process

6 Conclusion

In this paper, we have presented the parallelization of the EM algorithm applied to text categorization problem. We evaluated our parallel implementation on the 72 nodes PINUN Beowulf Cluster at Kasetsart University. The experimental results show reasonable speedups in our parallel implementation.

In future work, we will look at the improvement of the disk I/O. Our current implementation is based on the basic Unix I/O functions. When the problem sizes are scaled up, we require high performance I/O to reduce disk access costs. The parallel I/O defined the MPI-2 standard [16] is one solution. We expect that the parallel I/O will deliver much higher performance.

References

1. Agrawal, R., Shafer, J.C.: Parallel Mining of Association Rules. *IEEE Transaction on Knowledge and Data Engineering*. (1996)
2. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. The ACM Press, New York. (1999)
3. Dhillon, I.S., Modha, D.S.: A Data-Clustering Algorithm on Distributed Memory Multiprocessor. *Large-Scale Parallel Data Mining*. (1999) 245-260
4. Forman, G., Zhang, B.: Linear Speed-Up for a Parallel Non-Approximate Recasting of Center-Based Clustering Algorithms, including K-Means, K-Harmonic Means, and EM. *KDD Workshop on Distributed and Parallel Knowledge Discovery*. (2000)
5. Goharian, N., El-Ghazawi, T., Grossman, D., Chowdhury, A.: On the Enhancements Of a Sparse Matrix Information Retrieval Approach. *Proceedings of the*

- 1999 International Conference on Parallel and distributed Processing Techniques and Applications. (1999)
6. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: portable parallel programming with the message-passing. The MIT Press, Cambridge, MA (1999)
 7. Joachimes, T.: A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In proceedings of the Fourteenth International Conference on Machine Learning. (1997) 143-151
 8. Joshi, M.V., Karypis, G., Kumar, V.: ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining large Datasets. In Proceedings of International Parallel Processing Symposium. (1998)
 9. Lewis, D., Ringuette, M.: A Comparison of Two Learning Algorithms for Text Categorization. In Third Annual Symposium on Document Analysis and Information Retrieval. (1994) 81-93
 10. McCallum, A., Nigam, K.: A Comparison of Events Models for Naive Bayes Text Classification. Papers from the AAAI Workshop. (1998) 41-48
 11. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions. John Wiley & Sons (1997)
 12. Mitchell, T.: Machine Learning. McGraw-Hill, New York. (1997)
 13. Nigam, K., McCallum, A., Thrun, S., Mitchell, T.: Text Classification from Labeled and Unlabeled Documents using EM. Machine Learning. (2000) 103-134
 14. Nigam, K.: Using Unlabeled Data to Improve Text Classification. Doctoral Dissertation, Computer Science Department, Carnegie Mellon University, Technical Report CMU-CS- 01-126. (2001)
 15. Ruocco, A.S., Frieder, O.: Clustering and Classification of Large Document Bases in a Parallel Environment. JASIS 48(10). (1997) 932-943
 16. Thakur, R., Gropp, W., Lusk, E.: Optimizing Noncontiguous Accesses in MPI/IO. Parallel Computing. (2002) 83-105