Parallel Nearest Neighbour Algorithms for Text Categorization

Reynaldo Gil-García[1], José Manuel Badía-Contelles[2], and Aurora Pons-Porrata[1]

[1] Center of Pattern Recognition and Data Mining, Universidad de Oriente, Cuba
{gil,aurora}@csd.uo.edu.cu
[2] Dpt. Computer Science and Engineering, Universitat Jaume I, Castellón, Spain
badia@icc.uji.es

Abstract. In this paper we describe the parallelization of two nearest neighbour classification algorithms. Nearest neighbour methods are well-known machine learning techniques. They have been successfully applied to Text Categorization task. Based on standard parallel techniques we propose two versions of each algorithm on message passing architectures. We also include experimental results on a cluster of personal computers using a large text collection. Our algorithms attempt to balance the load among the processors, they are portable, and obtain very good speedups and scalability.

1 Introduction

Text Categorization (also known as text classification) is the task of assigning documents to one or more predefined categories. This task, which falls at the crossroads of Information Retrieval and Machine Learning, has witnessed a booming interest in the last years from researchers and developers alike. Text Categorization is an important component in many information management tasks such as spam detection [1], real time sorting of email or files into folders, document filtering [2], document dissemination, document routing [3], topic identification, classification of Web pages and automatic building of Yahoo!-style catalogs. Different learners have been applied in the Text Categorization literature, including probabilistic methods, decision tree and decision rule learners, example-based methods, support vector machines and classifier committees.

Most sequential text categorization algorithms have large running times. On the other hand, the volume of data available for analysis is growing rapidly. The huge size of text collections and their high dimension make classification a highly computationally-demanding application, to the point that parallel computing is an essential tool for its solution. Parallelism offers a natural and promising approach to cope with the problem of efficient categorization in large text collections.

Nearest neighbour (NN) classifiers are well-known methods for text categorization problems. This approach classifies an unknown sample into the categories of its nearest neighbours, according to some similarity measure. A particular case of NN classifiers is the *k*-nearest neighbour rule (*k*-NN), which assigns each unknown sample to the category most frequently represented among the *k* nearest training

A.-M. Kermarrec, L. Bougé, and T. Priol (Eds.): Euro-Par 2007, LNCS 4641, pp. 328–337, 2007.

samples. The NN classifiers include the following features: 1) conceptual simplicity, 2) easy implementation, 3) they can be designed even if there are few training samples, 4) known error rate bounds, 5) they can be implemented when categories are overlapped with each other, 6) good accuracy, 7) they have no design phase and simply store the training set, and 8) they can be performed in linear time with respect to the cardinality of the training set. The last feature can be a serious problem in very large text collections, because the computation of similarities is very time-consuming.

Several parallel versions of the NN learning method can already be found in the literature. Li [4] proposed several parallel nearest neighbour classification algorithms on two different types of SIMD computers. In [5] a parallel nearest neighbour classifier which uses attribute weights in the computation of distance is presented. This algorithm was implemented on the Connection Machine CM-2. It partitions the training set into p mutually exclusive data subsets that are distributed across the p processors. Then, each processor computes the similarity between the training samples in its local data subset and the new sample to be classified. Jin et al. [6] apply the same data distribution on a cluster parallel computer. Each node processes its training samples to calculate locally the k-nearest neighbours to the new sample and then, a global reduction is carried out to compute the overall k-nearest neighbours. The experimental results showed good speedup up to 8 nodes.

Finally, Jin et al. [7] also proposed a parallel version of k-NN algorithm on a shared memory computer. Each training sample is processed by one processor. After processing the sample, each processor updates the list of k current nearest neighbours. They used a full-replication scheme to avoid the race conditions. The obtained speedup was moderate up to four processors.

In this paper, we introduce scalable and efficient parallel versions of two nearest neighbour classification methods for distributed memory computers. Our versions use the standard parallel techniques: master-slave and pipelining. The algorithms were implemented using the MPI library on a cluster parallel computer. The performance of the proposed algorithms is evaluated on the Reuters Corpus Volume I [8], which is the new standard benchmark for text categorization tasks.

The remainder of this paper is organized as follows. In Section 2 we briefly review the nearest neighbour classification methods addressed in this paper. Section 3 describes our parallel algorithms. In Section 4 we show the experimental results and compare the proposed parallel versions. Finally, Section 5 contains a summary of the work.

2 Sequential Algorithms

As mentioned above, the nearest neighbour methods classify an unknown document d into the categories of its nearest neighbours in the training set. The training set is a collection of m documents labelled with their categories.

The nearest neighbour classifiers usually involve three phases: (i) the nearest neighbour finding from the training documents, (ii) a voting phase, in which each category assigns a vote to d, and (iii) a decision rule, in which a decision is made from these votes to classify the new document. During the last years, a large number of NN algorithms have aroused from various scientific communities. Many of them focus on

increasing classification rates, either changing the method to find the nearest neighbours or varying the voting scheme.

In this paper we focus on two NN methods, which assume different neighbourhood definitions. The first one, the traditional *k*-NN, starts at *d* and grows a spherical region until it encloses *k* training documents, where *k* is a user defined parameter.

The second method [9], considers a kind of neighbourhood which inspects a sufficiently small and near area to *d*. In this method, the number of neighbours is not fixed, but rather the neighbourhood radius is automatically adjusted from the nearest neighbour of *d*. This neighbourhood takes into account instead all neighbours enclosed within a spherical region defined from the nearest neighbour. The method uses two parameters α and β, which provide a convenient way of obtaining such a neighbourhood. From now on, we will refer to this method as *braNN*. In Figure 1 both neighbourhoods are graphically depicted. The shady regions represent the neighbourhoods in both methods.

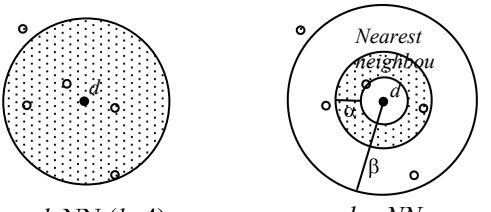


Fig. 1. Neighbourhoods of *k*-NN and *braNN* methods

Different methods have been used to calculate the votes of each category (second phase). One of these methods considers the similarity of the nearest neighbours and their category association to calculate the votes [10], i.e.,

$$V(c_i,d)=\frac{\sum_{d_j\in N(c_i)}\cos(d,d_j)}{\sum_{d_j\in N}\cos(d,d_j)}, \tag{1}$$

where N is the set of all nearest neighbours of the document d, $N(c_i)$ is the set of the nearest neighbours labelled with the category c_i, and $\cos(d,d_j)$ is the cosine between the two document vectors, which is the similarity measure commonly used in text categorization tasks.

From these category votes, several rules can be applied for deciding whether d should be classified under c_i (third phase). In this paper, we used the thresholding decision rule. According with this rule, the document is assigned to categories with the score greater than a certain threshold value γ. Notice that this decision rule allows a multi-label categorization.

Both nearest neighbour methods, *k*-NN and *braNN*, employ the same voting scheme and decision rule. To sum up, the steps of NN classifiers we use are shown in Algorithm 1. The step 3 depends on the neighbourhood definition. Algorithms 2 and 3

show the construction of the neighbourhood in *k*-NN and *braNN* methods, respectively.

Algorithm 1. NN classifiers

1. Let Tr be the training set.
2. Let d be the document to classify.
3. Build the set N of neighbours of d according with a certain neighbourhood definition.
4. Compute the votes $V(c_i,d)$ for each category c_i using formula (1).
5. For each category c_i:
   (a) If $V(c_i,d) \geq \gamma$:
      i. Assign d to c_i.

Algorithm 2. Construction of the *k*-NN neighbourhood

1. Let N be the list of neighbours of d, decreasingly ordered by its similarity with d.
2. $N = \varnothing$.
3. For each training document $d_j \in Tr$:
   (a) Insert d_j in N.
   (b) If $|N| > k$:
      i. Remove the last element of N.

Algorithm 3. Construction of the *braNN* neighbourhood

1. Build the set $N_\beta = \{d_j \in Tr / \cos(d,d_j) \geq \beta\}$
2. Let max be the similarity between d and its nearest neighbour in N_β.
3. Let N be the list of neighbours of d, $N = \varnothing$.
4. For each training document $d_j \in N_\beta$:
   (a) If $\cos(d,d_j) \geq max - \alpha$:
      i. Add d_j to N.

The space complexity of both algorithms is $O(m)$, because the whole training set is stored. On the other hand, its time complexity is $O(nm)$, where n is the number of documents to classify, because for each unknown sample the similarity with each training document is calculated.

3 Parallel Algorithms

Our parallel algorithms assume that we have p processors each with a local memory. These processors are connected using a communication network. We do not assume a specific interconnection topology for the communication network, but the access time to the local memory of each processor must be cheaper than time to communicate the same data with other processor.

We present two parallel versions of the algorithms *k*-NN and *braNN*. The first parallel algorithm uses a master-slave scheme whereas the second one uses the pipeline technique.

3.1 Master-Slave Algorithms

In the distributed memory setting, the nearest neighbour classifiers can be parallelized by dividing the data to be classified among the processors and replicating the training set. Thus, each processor can classify the data items it owns. A uniform data distribution can produce a load imbalance, since the computation of similarities of the sample to the training documents is the most time-consuming task, and the documents can have different sizes. An easy way to achieve load balancing is to use a master-slave scheme.

Our parallel algorithm can be described as follows. First, the master process broadcasts the training set. Then, the master sends a sample to each slave process. The slaves determine independently the nearest neighbours of their samples and classify them. Each time a slave finishes its work with a sample it asks for a new one to the master.

It is important to notice that no significant differences between *k*-NN and *braNN* exist in this parallel version. In the first case, each processor determines the *k* nearest neighbours (see Algorithm 2), whereas in the second one the processors determine the neighbourhood defined by α and β parameters (see Algorithm 3).

The time complexity of this scheme is $O(nm/(p\text{-}1))$, because the n unknown documents are classified independently by the p-1 slave processors and the communications and waiting time can be overlapped with the computations. The space complexity of this parallel algorithm is $O(m)$, because each processor stores the whole training set. Thus, this version does not have space scalability.

3.2 Pipeline Algorithms

Instead of dividing the data to be classified, in this parallel version, the training set is partitioned into p mutually exclusive data subsets that are distributed among the p processors. Once this data is distributed, the problem of finding the nearest neighbours of each document to classify can be broken up into several stages.

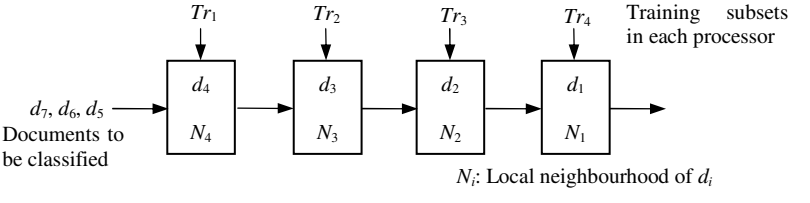


Fig. 2. Pipeline scheme

One pipeline solution could have a separate stage in each processor for updating the neighbourhood of each unknown sample, as shown in Figure 2. Each stage of the pipeline computes the similarities between the training documents in its local data subset and the document to be classified, and updates the list of nearest neighbours provided by the previous stage. Each unknown document and its partial neighbourhood

are sent from one processor to the next. Thus, the last processor obtains the global neighbourhood and it also classifies the sample.

The steps of this parallel algorithm are shown in Algorithm 4.

Algorithm 4. Pipelining

1. Scatter the training documents among the processors.
2. While True:
   (a) If processor 0:
      i. Load the unknown document *d*.
      ii. Build the local neighbourhood of *d*.
      iii. Send *d* and its neighbours to processor 1.
   (b) Else:
      i. Receive *d* and its neighbours from the previous processor.
      ii. Update the local neighbourhood of *d*.
      iii. If it is the last processor:
         A. Classify the document *d*.
      iv. Else:
         A. Send *d* and its neighbours to the next processor.

The neighbourhood updating (step 2(b)ii) depends on the nearest neighbour classifier. In parallel *k*-NN algorithm, each processor updates the neighbour list associated to each unknown sample it receives. This list is ordered by similarity to the sample and must only contain *k* documents. When a processor receives the sample and the local list of neighbours from the previous one, it inserts its training documents in the list according with its similarity to the sample.

On the other hand, in parallel *braNN* algorithm, each processor updates the nearest neighbour to the unknown sample and the list of those training documents whose similarity is greater than both β and *max*-α (*max* is the similarity between the nearest neighbour and the new sample).

The time complexity is $O(nm/p)$, because each processor computes the similarities between each sample and its m/p training documents. In this parallel algorithm the communications can be overlapped with the computations. The space complexity of this parallel version in both algorithms is $O(m/p)$, because the training set is scattered among the p processors. Thus, this parallel algorithm has space scalability.

Notice that both, this parallel version and the algorithm proposed by Jin [6], distribute the training set among the processors and that local neighbourhoods are calculated in each processor. However, both algorithms differ on the procedure to compute the global neighbourhood. In Jin's algorithm a global reduction is carried out whereas we use a pipeline.

4 Performance Evaluation

The target platform for our experimental analysis is a cluster of SMP processors connected through an Infinitband network. The cluster consists of 8 nodes, each containing 4 Intel Itanium-2 processors and 4 Gbyte of RAM. The proposed parallel algorithms have been implemented on a Linux operating system, and we have used a specific implementation of the MPI message-passing library that offers small latencies and high bandwidths on the Infinitband network. This library also optimizes

the communications between processors on the same node by exploiting the shared memory as communication mechanism. We have executed the parallel algorithms varying the number of processors from 4 to 32.

We used the Reuters Corpus Volume I (RCV1-v2) [8] as our testing medium. RCV1-v2 collection consists of over 800000 newswire stories that have been manually classified in 103 categories. This collection is partitioned (according to the LYRL2004 split we have adopted) into a training set of 23149 documents and a test set of 781265 documents. The documents are represented using the traditional vector-space model. Terms are statistically weighted using Cornell *ltc* term weighting [11].

Due to the fact that we focus on a parallelization scheme, no accuracy results for the data are given in this paper. We neither tune the parameters of the algorithms, but we fix the values that report good results in the literature.

In this paper, we compare our two parallel versions of *k*-NN and *braNN* algorithms. We also implemented the parallel version proposed by Jin [6] and compare it with them. The performance comparison using different number of processors for *k*-NN and *braNN* algorithms is shown in Figures 3 and 4, respectively.

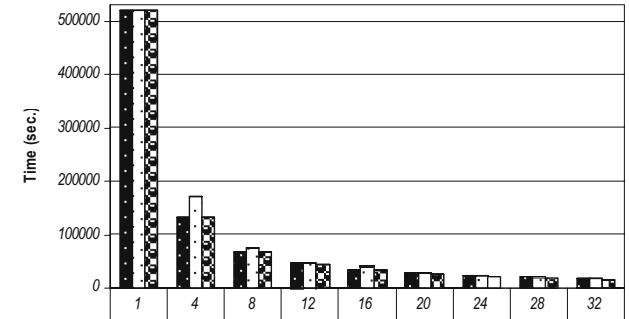
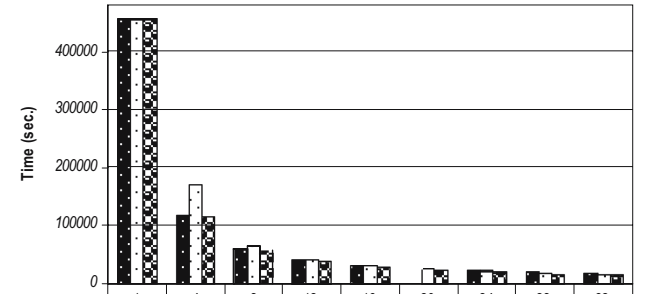


Fig. 3. Running times of parallel *k*-NN versions

Several observations can be made by analyzing the results in these figures. First, both sequential algorithms spent a lot of time classifying the documents. Second, all parallel versions clearly reduce the sequential time. Notice that the sequential *k*-NN algorithm takes about 6 days to classify this collection, while the pipeline parallel version reduces this time to 4.08 hours on 32 processors. In case of *braNN* algorithm, the time decreases from 5.26 days down to 3.58 hours.

A third observation that also emerges clearly from the figures is that all parallel versions of *braNN* algorithm are less time-consuming than the corresponding parallel versions of *k*-NN algorithm independently of the number of processors used. This fact is in agreement with the results presented earlier in [9]. This can be explained because *braNN* algorithm does not need to create a sorted list of *k* nearest neighbours.



	1	4	8	12	16	20	24	28	32
Jin	455196	118348	59217	40178	30912	24250	20386	17762	15946
Master-Slave	455196	170905	64282	41389	30039	23760	19897	16704	14540
Pipelining	455196	115005	58175	37943	27862	21635	17638	14719	12887

Number of processors

Fig. 4. Running times of parallel *braNN* versions

Figure 5 shows the speedups obtained by the parallel *k*-NN algorithms. As we can see, Jin's algorithm slightly outperforms our parallel version that uses the master-slave model for a small number of processors. However, our algorithm obtains larger speedups while increasing the number of processors. This can be explained because our algorithm uses only *p*-1 processors to classify the documents and, therefore, its speedup is always near *p*-1. On the contrary, as Jin's algorithm uses a global reduction operation, its efficiency goes down as the number of processors increases. Thus, we can conclude that our parallel version is more scalable than Jin's algorithm.

On the other hand, Figure 5 shows clearly that pipeline parallel version achieves superlinear speedup. We think that it is due to the data distribution used. The higher

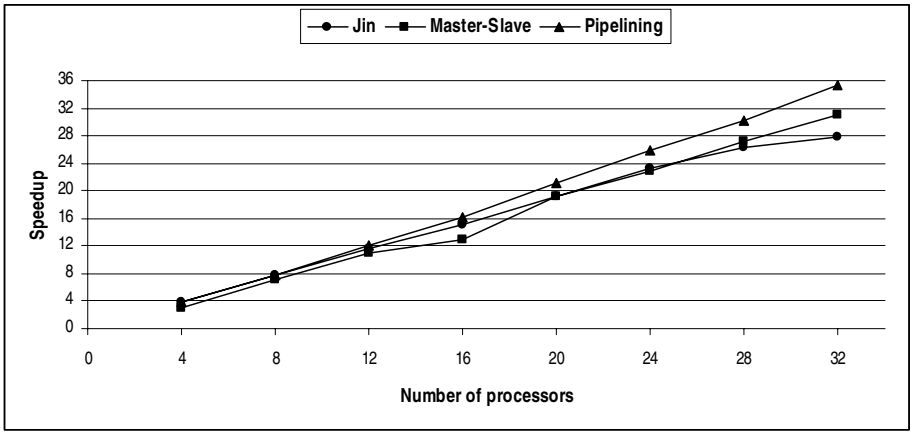


Fig. 5. Speedup of parallel *k*-NN versions

the number of processors, the lower the problem size per processor is, and therefore, the cache hit ratio is increased. The overlapping between the communications and the computations also contributes positively to this result.

Finally, the speedups obtained by the parallel versions of *braNN* algorithm are shown in Figure 6. As it can be noticed, the results are very similar to those obtained by *k*-NN parallel algorithms. The parallel version that uses a pipeline model also achieves the best speedups in this case.

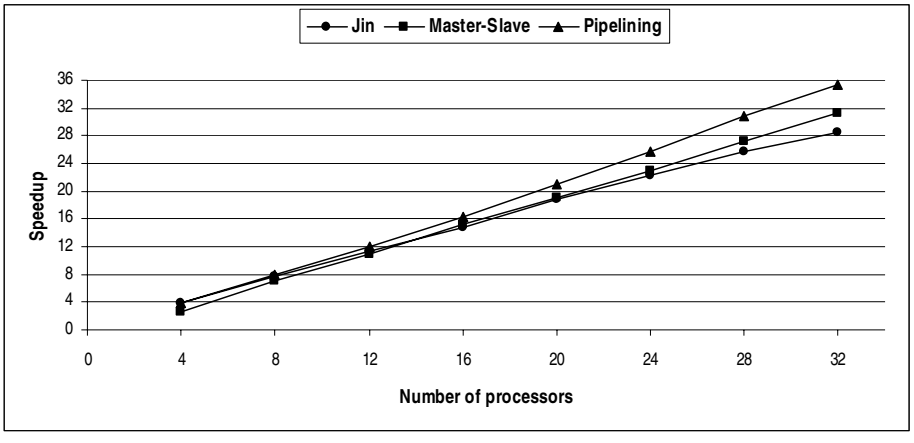


Fig. 6. Speedup of parallel *braNN* versions

5 Conclusion

In this paper, we have focused on distributed memory parallelization of nearest neighbour classifiers. Two new parallel versions of these classifiers based on standard techniques: master-slave model and pipeline scheme are proposed. The resulting algorithms are portable, because they are based on standard tools, including the MPI message-passing library.

We have reported experimental results on RCV1-v2 corpus, which is the new standard benchmark for text categorization tasks. These experiments establish the following:

1. Sequential nearest neighbour classification requires intensive computation on very large text collections.
2. Very good speedup is achieved for each of the proposed parallel algorithms.
3. In the comparison between our parallel approach that uses a master-slave model and Jin's algorithm, we conclude that despite its poor space scalability, our algorithm obtains better time scalability.
4. The parallel algorithm that uses a pipeline model offers clear advantages over the parallel versions we experimented with in that it is both spatially and temporally scalable. This method achieves the same space scalability as Jin's

algorithm and the best time scalability. The pipeline version also obtains superlinear speedup.

The proposed parallel algorithms can be used in many highly computationally demanding-applications such as information organization, filtering, routing, topic tracking and text categorization tasks, allowing to process huge text collections.

Acknowledgments. This work was partially supported by the Research Promotion Program 2006 of Universitat Jaume I, Spain, the CICYT project TIN2005-09037-C02-02 and FEDER.

References

[1] Drucker, H., Wu, D., Vapnik, V.N.: Support Vector Machines for Spam Categorization. IEEE Transactions on Neural Networks 10(5), 1048–1054 (1999)
[2] Eichmann, D., Srinivasan, P.: Adaptive Filtering of Newswire Stories using Two-Level Clustering. Information Retrieval 5, 209–237 (2002)
[3] Iyer, R.D., Lewis, D.D., Schapire, R.E., Singer, Y., Singhal, A.: Boosting for Document Routing. In: Proceedings of the Ninth International Conference on Information and Knowledge Management (2000)
[4] Li, X.: Nearest neighbor classification on two types of SIMD machines. Parallel Computing 17, 381–407 (1991)
[5] Creecy, R.H., Masand, B.M., Smith, S.J., Waltz, D.L.: Trading MIPS and memory for knowledge engineering. Comm. of the ACM 35(8), 48–63 (1992)
[6] Jin, R., Agrawal, G.: A Middleware for Developing Parallel Data Mining Implementations. In: Proceedings of the First SIAM Conference on Data Mining (2001)
[7] Jin, R., Yang, G., Agrawal, G.: Shared memory parallelization of data mining algorithms: Techniques, programming interface and performance. IEEE Transactions on Knowledge and Data Engineering 17, 71–89 (2005)
[8] Lewis, D., Yang, Y., Rose, T., Li, F.: Rcv1: A new benchmark collection for text categorization research. Machine Learning Reseach 5, 361–397 (2004)
[9] Gil-García, R., Pons-Porrata, A.: A new nearest neighbor rule for text categorization. In: Martínez-Trinidad, J.F., Carrasco Ochoa, J.A., Kittler, J. (eds.) CIARP 2006. LNCS, vol. 4225, pp. 814–823. Springer, Heidelberg (2006)
[10] Yang, Y.: Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In: SIGIR'94, 17th ACM International Conference on Research and Development in Information Retrieval, Ireland, pp. 13–22. ACM Press, New York (1994)
[11] Buckley, C., Salton, G., Allan, J.: The effect of adding relevance information in a relevance feedback environment. In: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp. 292–300. ACM Press, New York (1994)