


Acceleration of a Feature Selection Algorithm Using High Performance Computing [†]

Bieito Beceiro * , Jorge González-Domínguez  and Juan Touriño 

Computer Architecture Group, CITIC, Universidade da Coruña, 15071 A Coruña, Spain; jgonzalezd@udc.es (J.G.-D.); juan@udc.es (J.T.)

* Correspondence: bieito.beceiro.fernandez@udc.es

† Presented at the 3rd XoveTIC Conference, A Coruña, Spain, 8–9 October 2020.

Published: 1 September 2020



Abstract: Feature selection is a subfield of data analysis that is on reducing the dimensionality of datasets, so that subsequent analyses over them can be performed in affordable execution times while keeping the same results. Joint Mutual Information (JMI) is a highly used feature selection method that removes irrelevant and redundant characteristics. Nevertheless, it has high computational complexity. In this work, we present a multithreaded MPI parallel implementation of JMI to accelerate its execution on distributed memory systems, reaching speedups of up to 198.60 when running on 256 cores, and allowing for the analysis of very large datasets that do not fit in the main memory of a single node.

Keywords: feature selection; mutual information; machine learning; high performance computing; MPI

1. Introduction

Feature selection algorithms are data analytics techniques that aim to reduce the size of large datasets by detecting those features that are relevant and discarding the irrelevant ones. These methods have become extremely popular in the last years due to the increasing amount of data that are gathered every day and need to be processed. Moreover, the algorithms used to process data are usually computationally expensive, so the execution times are excessive when datasets grow in size.

In this work, we focus on the Joint Mutual Information (JMI) algorithm [1], since it provides the best tradeoff in terms of accuracy, stability, and flexibility for datasets of various properties, as stated in [2]. JMI follows a filter approach for feature selection, which is, it is used as an independent task performed before the subsequent machine learning algorithm. JMI receives from the user the number of features to select, and it iteratively takes the feature with the highest score, calculated as a tradeoff between its relevance and redundancy. Despite its good results, the main drawback of JMI is its quadratic complexity that prevents its usage to analyze large datasets.

The aim of this work is the development of a new version of the JMI algorithm that is faster than the state-of-the-art solution: the C implementation included in the widely employed and cited suite FEAST [2]. Our implementation must also enable the analysis of huge datasets that do not fit in the memory of conventional computing systems. Both of the goals have been achieved thanks to a novel multithreaded MPI implementation of JMI that can be executed on distributed memory systems, so that it can take advantage of enabling parallel work with several nodes and, thus, reduce the execution time. In addition, this parallel version of JMI is able to divide and distribute extremely large datasets among the memory of all nodes, making it possible to process them.

2. Parallel Implementation of JMI

JMI works with datasets that are represented as matrices, where the rows are the features and the columns are the samples. The output dataset is formatted with a pair for each selected feature. The first element of the pair is its position in the original dataset, while the second one is the score. In this work, we propose a hybrid parallel approach of JMI based on the MPI message-passing library and C++ threads that works with the same input/output format than FEAST and guarantees the same exact results, but with significantly reduced execution times. This has been achieved by using a feature-wise domain decomposition, which is, the features of the input dataset are divided into blocks of the same size so that each block is processed in parallel by a different processing element. Our implementation makes use of this domain decomposition at two levels of parallelism. First, the dataset is distributed among MPI processes. Besides the execution time gains, this MPI parallelization makes it possible the joint usage of the memory of several nodes. Second, the workload of each process is distributed among threads.

Furthermore, two outstanding performance optimizations have been applied to our parallel JMI implementation:

- Semi-distributed data loading to take advantage of the sparse storage format used by the input datasets. While, in a naive data load implementation, one process would read the whole dataset and would scatter it among the other processes, in our semi-distributed approach the root process loads the dataset in sparse format (which fits in memory due to its reduced size), broadcasts it to all of the other processes and then each process transforms the portion of data that needs to work with into a dense matrix.
- Range compression. This is a data preprocessing technique that can reduce the overall execution time and memory usage. It is based on renaming the values of the features so that the range of the values becomes minimal, which has no effect on the results since scores are calculated by value co-occurrences and not by the values themselves. This technique significantly accelerates the creation of the two-dimensional (2D)-histogram that is needed to count co-occurrences between two features. Without range compression, there will be some rows and/or columns that will not affect the result (specifically, those ones created for values that do not appear in the feature), but they are included in the histogram and also have to be processed, increasing the memory consumption and execution time. However, this is avoided when applying our range compression technique, since it assures that only the rows and columns that are needed for the score calculation are included in the histogram. Moreover, this technique introduces almost no overhead due to its linear computational complexity and the possibility of being executed in parallel.

3. Results and Conclusions

The experiments for the analysis of our optimized parallel version of JMI were conducted on 16 nodes of the “Pluton” cluster, a distributed memory system that is based on Intel Xeon processors installed at CITIC. Each node is composed of two processors with eight cores each (16 logical threads using HyperThreading) and 64 GB of main memory. The whole system provides a total of 256 cores (512 logical threads with HyperThreading) and 1 TB of memory. In order to keep the reproducibility of the experiments, we have used five representative datasets that are publicly available in the LibSVM website (<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>). Their properties are shown at the left half of Table 1.

Table 1 also shows the runtime and speedup (in parentheses) for different number of nodes. Regarding the parallel executions, the same configuration for each node was used, as it proved to be the most efficient one after some preliminary tests: two MPI processes with eight cores each (16 logical threads per MPI process using HyperThreading). However, due to memory problems for the largest datasets (SVHN and E2006), we had to use one MPI process with 16 cores each (32 logical threads) in these cases. Moreover, E2006 is so large (~512 GB as a matrix in memory) that it does not fit in

less than 16 nodes, so it could be executed only using this configuration. As can be seen, the parallel version of JMI performs well and offers excellent scalability (execution times largely decrease when the number of cores increases). Furthermore, it enables the analysis of huge datasets (E2006) that the original version of JMI could not handle due to their size.

Table 1. Dataset properties, execution times (in seconds), and speedups (in parentheses) to select 200 features with JMI.

Dataset	Features	Samples	Original JMI	1 Node	2 Nodes	4 Nodes	8 Nodes	16 Nodes
Epsilon	2000	400,000	1837.59	108.29 (16.97)	57.88 (31.75)	30.86 (59.55)	18.49 (99.38)	12.03 (152.75)
RCV1	47,236	20,242	1834.87	94.86 (19.34)	48.77 (37.62)	27.10 (67.71)	16.38 (112.02)	10.70 (171.48)
News20	62,061	15,935	1813.70	96.38 (18.82)	49.51 (36.63)	27.30 (66.44)	16.12 (112.51)	10.64 (170.46)
SVHN	3072	531,131	8132.46	424.98 (19.14)	264.40 (30.76)	142.95 (56.89)	74.04 (109.84)	40.95 (198.60)
E2006	4,272,227	16,087	–	–	–	–	–	787.64

Author Contributions: Conceptualization, J.G.-D. and J.T.; methodology, B.B., J.G.-D. and J.T.; implementation, B.B.; validation, B.B.; writing—original draft preparation, B.B.; writing—review and editing, J.G.-D. and J.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00/AEI/10.13039/501100011033), and by Xunta de Galicia and FEDER funds of the EU (Centro de Investigación de Galicia accreditation 2019-2022, ref. ED431G2019/01; Consolidation Program of Competitive Reference Groups, ED431C 2017/04).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yang, H.H.; Moody, J. Data Visualization and Feature Selection: New Algorithms for Nongaussian Data. In *Proceedings of the NIPS’99 12th International Conference on Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 1999; pp. 687–693.
2. Brown, G.; Pocock, A.; Zhao, M.J.; Luján, M. Conditional Likelihood Maximisation: A Unifying Framework for Information Theoretic Feature Selection. *J. Mach. Learn. Res.* **2012**, *13*, 27–66.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).