Neurocomputing 191 (2016) 175–186

Contents lists available at ScienceDirect

Neurocomputing

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

Efficient parallel implementation of kernel methods ☆

Roberto Díaz-Morales *, Ángel Navia-Vázquez

DTSC, Univ. Carlos III de Madrid, Avda Universidad 30, 28911 Leganés, Madrid, Spain

ARTICLE INFO

Article history:
Received 3 March 2015
Received in revised form
18 June 2015
Accepted 29 November 2015
Communicated by Zheng Wenming
Available online 3 February 2016

Keywords:
Support Vector Machine
Gaussian Process
OpenMP
Parallel
Kernel methods
Matrix inversion

ABSTRACT

The availability of multi-core processors has motivated an increasing interest in research lines about parallelization of machine learning algorithms. Kernel methods such as Support Vector Machines (SVMs) or Gaussian Processes (GPs), in spite of their efficacy solving problems of classification and regression, have a very high computational cost and usually produce very large models. In this paper we present parallel algorithmic implementations of Semiparametric SVM (Parallel Semiparametric SVM, PS-SVM) and Gaussian Processes (Parallel full GP, P-GP and Parallel Semiparametric GP, PS-GP). We have implemented the proposed methods using OpenMP and benchmarked them against other state of the art methods, showing their good performance and advantages in both computation time and final model size.

1. Introduction

Kernel methods are very popular in machine learning because they produce highly competitive results in many practical tasks. They transform the input space onto a high dimensional one where inner products are computed using a kernel function. The most relevant techniques are Support Vector Machines (SVMs) for classification problems and Gaussian Processes (GPs) for regression.

Support Vector Machines [1] are one of the most successful kernel techniques, which aim to obtain a maximum margin separating hyper-plane. They are very popular because they automatically adjust the machine size and also produce highly competitive results in many real world problems. The resulting size of the classifier is often very large, that represents a high computational cost. Many research lines have emerged to solve this problem of complexity and scalability. Some works [2–5] calculate a full SVM and reduce afterwards the machine size by solving a preimage problem [6]. In [7,8], to avoid the calculation of a full SVM, they propose an iterative growing architecture. In [8], Sparse Greedy Matrix Approximation (SGMA) is proposed to iteratively select candidates to grow a semiparametric model. Peng et al. [9] introduce a criterion for identification of support vectors leading to a reduced support vector set. Other works [10] are focused on improving the classification complexity using decision trees.

Gaussian processes [11] are also non-parametric methods considered the "state-of-art" solving regression problems and relying on probabilistic Bayesian models. Unfortunately, their direct application is limited due to the high training time and computational cost $O(n^3)$ in non-sparse solutions, where n is the size of the training set. There are also some iterative greedy schemes that obtain a reduced GP. Among those, [12,13] are based on minimizing Kullback–Leibler divergences, [14] uses a MAP criteria to select in every iteration the candidate to grow the model and [15] selects in every iteration the element that maximizes the evidence to avoid overfitting problems.

Since run time is the main problem of kernel methods, parallelization is one of the most important techniques to accelerate them. Currently, the semiconductor industry is creating new designs of processors that increase their performance with the inclusion of more cores in a single chip. With the emergence of multi-core processors and new programming interfaces such as OpenMP [16] to develop parallel software, many research lines about parallelization in kernel methods have been opened.

Early works on parallelization in SVMs propose to split the training set, train different SVMs on every data chunk and combine the results using a neural network [17] or to train a new SVM using the obtained Support Vectors [18]. In [19], a parallel version using a cascade of SVMs is used. Recently new methods have appeared, such as PSVM [20], Parallel SMO [21,22] or Graphics Processing Unit (GPU) Tailored Approach SVM [23]. After the apparition of the Big Data technologies a MapReduced based SVM is used in [24,25] to solve problems in a distributed environment.

☆This work has been partly supported by Spanish MEC project TIN2011-24533.
* Corresponding author. Tel.: +34 624 6256.
E-mail address: rdiazm@tsc.uc3m.es (R. Díaz-Morales).

http://dx.doi.org/10.1016/j.neucom.2015.11.097
0925-2312/

PSVM solves the Quadratic Programming problem using a parallel implementation of the Interior Point Method (IPM) [26] and Incomplete Cholesky Factorization. Parallel SMO uses a parallel version of SMO [27] that divides the quadratic problem into a series of smaller subproblems, which can be solved analytically. An implementation for GPUs that uses clustering techniques to handle sparse data sets is presented in [23]. In GPs [28] uses domain decomposition to solve 2-dimensional problems in parallel.

Due to the fact that the run time of the training procedure and the complexity of the model are the main weaknesses of Kernel methods, our proposal here consists in the development of new schemas that can address these issues. To that end we are benefiting from two different techniques:

Semiparametric models: That can solve the issue of the model complexity, as presented in previous works [29], because the final machines are written as a function of a set of representatives, instead of support Vectors (as in SVMs) or all data (as in GPs). These models have been shown to achieve similar performance as the full machines but with a lower computational cost and complexity.

Parallel computing: That can solve the issue of the scalability and the excessive run time of the training procedure by simultaneously using multiple computer resources to solve the problem.

By using these techniques we have developed three different models.

- *PS-SVM* : A parallel and semiparametric version of the SVM.
- *P-GP*: A parallel version of the GPs.
- *PS-GP*: A parallel and semiparametric version of the GPs.

This paper is organized as follows. In Section 2 we describe our algorithms. Experimental results are provided in Section 3. Finally we describe the conclusions in Section 4.

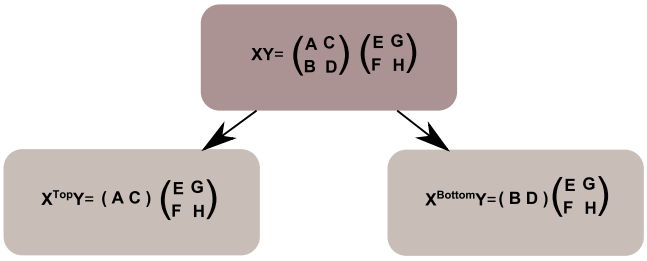

Fig. 1. Division of a matrix product into two subtasks.

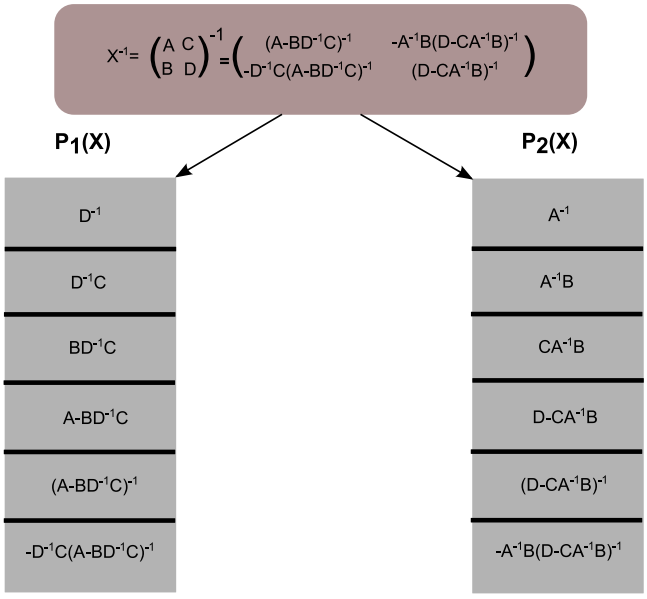

Fig. 2. Division of matrix inversion into two subtasks.

2. Algorithms

When developing parallel code, the two most important issues to avoid if possible are:

- *Non-parallelizable sections of code*: Because they put the upper bound of speedup in our model according to Amdahl's law [30]. The run time of our non-parallel code is absolutely despicable comparing to the whole run time.
- *Communication between threads*: To avoid possible bottlenecks we have selected OpenMP as the parallel framework because when a subtask finishes its job another subtask can access the results stored in the RAM memory.

2.1. Parallelization of basic operations

2.1.1. Products

In a shared memory environment, the parallelization of products of matrices can be done in an efficient way, as seen in [31] by sharing the rows of the resulting matrix among the different cores.

This schema is easily implementable with any parallel programming interface as OpenMP. In Fig. 1 $\mathbf{X^{Top}}$ represents the half top rows of $\mathbf{X}$ and $\mathbf{X^{Bottom}}$ the half bottom rows, i.e., $\mathbf{X^{Top}} = (\mathbf{AC})$, $\mathbf{X^{Bottom}} = (\mathbf{BD})$.

2.1.2. Inversions

Traditional methods to invert matrices are sequential, so their parallelization is impossible. To perform an inversion in parallel we have divided the matrix into four submatrices (quadtree) and used the block matrix pseudoinversion to divide the process into two subtasks (P1 and P2). Fig. 2 shows the parallelization of a matrix inversion for two cores.

Each subtask is composed of two inversions and three products, the run time is slightly higher than half of the whole process. This has been partially solved using LU inversion and back substitution, because it is possible to implement an inversion and a product $\mathbf{A^{-1}B}$ with the same computational cost than just an inversion $\mathbf{A^{-1}}$ [32].

Each subtask is composed of products and inversions that can be recursively divided again until all the cores in the system are used. For example, in Fig. 3 the division when four cores are used is shown.

As we increase the number of cores and tasks, the run time of every subtask decreases. For this approach to be useful, the time spent in communications between tasks should be much lower than the runtime of every task, to avoid bottlenecks.

2.1.3. Greedy techniques

In kernel methods, greedy algorithms [5,14,15,33] are one of the most common techniques to build a semiparametric model. These algorithms are proposed for iterative growing architectures, they are based on selecting a group of candidates $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ from the training set, evaluate them using an objective function $f(\mathbf{x}_i)$ and updating the architecture using the element that has obtained the best result.

To parallelize these algorithms, the elements have been distributed among the available cores in the system, the schema is represented in Fig. 4 and can be easily implemented using any parallelization technique.