Fusion Engineering and Design 129 (2018) 130–133

Contents lists available at ScienceDirect

Fusion Engineering and Design

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

Software parallelization of a probabilistic classifier based on Venn Prediction: Application to the TJ-II Thomson Scattering

F.J. Martínez[a], J. Vega[b], S. Dormido-Canto[a], I. Pastor[b], E. Fabregas[a,*], G. Farias[c]

[a] Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain
[b] Laboratorio Nacional de Fusión, CIEMAT, Avda. Complutense, 40, 28040 Madrid, Spain
[c] Pontificia Universidad Catolica de Valparaiso, Avda. Brasil 2147, Valparaiso, Chile

ARTICLE INFO

Keywords:
Venn Predictors
Parallelization
Probabilistic classifier
Thomson Scattering

ABSTRACT

One of the recurring problems encountered in the development of automatic classification problems is the so-called “curse of dimensionality”. Procedures that are computationally manageable in low dimensional spaces can become unfeasible in spaces of hundreds of dimensions due to the need of long computational times. This paper shows the parallelization of a probabilistic classifier based on Venn Predictors (VP). VP determine a probability interval to qualify how accurate and reliable each individual classification is. The parallelized code has been applied to the classification of the images from the CCD camera of the TJ-II Thomson Scattering. The aver- age probability and probability interval are a very efficient prediction from the prediction perspective.

1. Introduction

TJ-II discharges generate plasmas for a maximum time of 0.35 s and therefore, collect a large amount of information. It is necessary to have fast and effective classifiers due to the large amount of collected data. When the performance of classifiers for data processing needs to be increased, the use of high performance computing (HPC) [1] could be a potential solution. Parallel processing involves many methods that go from parallel architectures and parallel algorithms to parallel programming languages and performance analysis, to mention but a few.

The curse of dimensionality, also known as Hughes effect, refers to various phenomena that occur when large numbers of samples have to be processed. These phenomena do not occur when working in low dimensions spaces. When the space dimensions increase, the space volume increases too and it does so very rapidly. The curse of dimensionality together with overfitting are the two main problems linked to the classification tasks. The phenomenon argues that if a fixed number of samples are available, then the predictive capacity of a classification algorithm decreases when the dimensions increase. In this sense, for classification problems, increasing the number of dimensions is equivalent to increasing the number of characteristics of the vectors. It is easy to think that a greater number of characteristics will suppose more information and, therefore, a better prediction. The fact is that the curse of dimensionality indicates the opposite: the importance is not in the characteristics quantity but in their relevance in the classification. In a learning algorithm, there may be two limitations such as the number of training samples and the computational time. When a large scale classification problem is addressed, in some cases the samples handled are in the order of millions with a variety of different dimensions.

In the area of machine learning, the results improve if the system is able to calculate a reliability index associated with prediction. In some areas such as medical diagnosis it is very important to have a quality index of the prediction. For determining this index it is necessary to apply Venn Predictors (VP) [2]. The principal advantage of VP is to qualify the predictions with a probability interval. This measure is not just a probability value as in a Bayesian classifier. This can be seen as a probability error bar for the prediction. In addition to this, VP are well calibrated which means that the probability is related to a frequentist justification from the data. The only assumption required for VP is that the data distribution is an independent identical distribution (i.i.d. assumption). However, to achieve their goal the VP need many more computational operations.

There are many algorithms of automatic learning that allow making estimations classification. However, many of them lack a measure of confidence to evaluate the error in the prediction. For a new example, it is necessary to have an algorithm that provides a probability distribution for each possible label as VP do.

The classification is satisfied calculating the probability of each of the possible classes involved in the calculation. The calculations obviously grow and make the VP computationally inefficient. Fig. 1 shows the increase in the number of operations while using VP when the

* Corresponding author.
E-mail address: efabregas@bec.uned.es (E. Fabregas).

https://doi.org/10.1016/j.fusengdes.2018.01.066
Received 31 May 2017; Received in revised form 26 December 2017; Accepted 26 January 2018

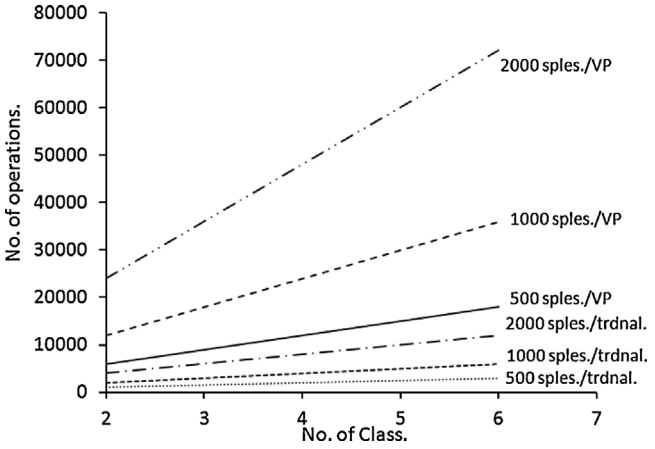

Fig. 1. Increase of calculations.

number of different classes and different samples increases as well.

This predictor characteristic is originated by the underlying algorithms. This work focuses on the VP parallelization to minimize the impact of the increase in the number of operations. The main contributions of this work can be summarized as follows: (a) Study and adaptation of three parallelization algorithms; (b) Application of parallelism to probabilistic Venn Predictors.

The rest of the paper is structured as follows. Section 2 introduces the Venn Predictors and underlying algorithms applied. Section 3 shows the algorithms parallelized. Section 4 presents the parallel implementation and experimental sets. Section 5 details the results obtained and summarizes the main conclusions.

2. Venn Predictors and underlying algorithms

To introduce Venn Predictors [3], we begin with a training set of the form $\{z_1, z_2, \ldots, z_{n-1}\}$, where each $z_i \in Z$ is a pair (x_i,y_i) where x_i is an object and y_i is a label. For a new object x_n, we intend to estimate its probability of belonging to each class $Y_j \in \{Y_1, Y_2, \ldots, Y_c\}$. Venn Predictors assign each of the possible classifications Y_j to x_n and divide all examples $\{(x_1, y_1), \ldots, (x_n, Y_j)\}$ into a number of categories based on a *taxonomy*. A *taxonomy* is a sequence A_n, $n = 1, \ldots, N$ of finite measurable partitions of the space $Z^{(n)} \times Z$, where $Z^{(n)}$ is the set of all multisets of elements of Z of length n. We will write $A_n(\{z_1, \ldots, z_n\}, z_i)$ for the category of the partition A_n that contains $(\{z_1, \ldots, z_n\}, z_i)$. Every *taxonomy* $A_1, A_2, \ldots, A_N$ defines a different VP.

Practically any known machine learning algorithm are called *underlying algorithms*. The VP can be considered as environment algorithms rather than simple predictors. Reference [2] indicates that, in general, a Venn Predictor can be developed in a top layer of any learning algorithm. In this work the support vector machines algorithm (SVM) has been used as underlying algorithm and therefore our *taxonomy* is based on SVM [4].

SVM [5] provides a machine learning algorithm for a binary classification problem. SVM can be extended in an easy way to multiclassification problems (for example an approach one-versus-the-rest). The main idea of SVM is that it maps the data into such a feature space where the classes are linearly separable. The separating hyperplane is optimal in the sense that it maximizes the distance from the closest data points belonging to both classes, which are the support vectors. The optimal hyperplane is usually found by solving a quadratic programming (QP) problem which is usually quite complex, time consuming and prone to numerical instabilities.

SVM uses supervised learning. This means that the training of a SVM classifier requires a pre-labelled data-set which is often referred to as a training set. The nonlinear mapping can be done by using kernels. Kernels define the measure of similarity and they are essential to SVM. The type of kernel function can thereby be crucial for classification results. For image data classification, higher order kernels, such as radial basis functions (RBF) and polynomial kernels, are most widely used. In this work the RBF kernel has been used.

Next section addresses the parallelization of three methods that implement SVM. These methods are independent from each other and the parallelization has been carried out separately, because a combined solution of three is not possible. These three methods are cascade SVM, Spread-Kernel SVM and Kernel Adatron SVM.

3. Paralellization

The **cascade SVM (cSVM)** algorithm [6,7] is a parallelization concept that optimizes sub-problems independently and iteratively combines the support vectors. This algorithm formalizes a parallelization at the level of training data. Before continuing, the concept of *worker* must be introduced. The *workers* are processes created in the system for certain tasks and their execution is controlled. Other algorithms distribute the kernel matrix among the *workers*, but here it is necessary to divide the input data between the number of available *workers*. In such a way, a portion of the training data is assigned to each worker. The dependence of the different steps of the algorithm makes it difficult to gain computational speed. For this reason the input set is divided into smaller sets for building a network where the *workers* work collaboratively.

At each level the vectors of the preceding level are combined to leave only one set of them at the last level. The result of the last level feeds again the input of the first level. In the first level the *workers* receive as input all the support vectors (SV) calculated from the last level. Convergence for this method is achieved when all the *workers* of the first level do not have a support vector to add. In this algorithm the parallelization of the problem is performed in the distribution of training data.

The method can be as follows: (1) Partition the data into k disjoint subsets of equal size; (2) Independently, train a SVM on each of the data subsets; (3) Combine the SVs output of each pair of SVM which will be the input to the next SVM level; (4) Repeat steps 2 and 3 until the first level does not add more SVs.

Spread-Kernel SVM (skSVM) algorithm [8] searches convergence through parallelizing the working set calculation. To achieve this objective the kernel matrix must be distributed among all workers that make up the cluster. There are two ways to load in memory the part of the kernel matrix that corresponds to each worker. One of them is to have it loaded in memory and once the parallel session has been started, make the corresponding distribution. The second way would be to assign the proportional data to each worker and assign the minimum and maximum indexes of each partition to each worker. To avoid managing all updates from the frontend worker, each worker sends its data to the adjacent worker (in a ring structure) until reaching the frontend worker. With this method there is no waste of time in communications. This algorithm performs the following steps: (1) Assign upper and lower indexes; (2) Calculate working-set; (3) Calculate vector alphas; (4) Update gradient array; (5) Repeat steps 2, 3 and 4 until convergence.

In the **kernel-Adatron SVM (kaSVM)** the algorithm parallelized is the one proposed in [9] where it becomes a reformulation of the bias. The previous algorithms are based on working set calculation to reach convergence, however the solution of the problem is parallelized differently. Here the optimization problem is resolved iteratively where each *worker* calculates the solution with its part of the kernel array. That is, each *worker* works with a section of the kernel array and sends its update to the *worker* manager. The *worker* manager is responsible for receiving all the updates and forwarding them back to the rest of *workers*. There are three problems associated with this algorithm: the high management rate that has the main *worker*, the time spent on sending data, and finally, the limitations of memory. Unlike the previous algorithm where it was not necessary to have the kernel array