Swarm and Evolutionary Computation 26 (2016) 50–55

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Regular Paper

Towards power plant output modelling and optimization using parallel Regression Random Forest

Jan Janoušek, Petr Gajdoš, Pavel Dohnálek*, Michal Radecký

Department of Computer Science, FEECS VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 29 October 2014
Received in revised form
16 July 2015
Accepted 22 July 2015
Available online 14 August 2015

Keywords:
Random Forests
Regression
GPU
CUDA
Parallel computing

ABSTRACT

In this paper, we explore the possibilities of using the Random Forest algorithm in its regression version to predict the power output of a power plant based on hourly measured data. This is a task commonly leading to a optimization problem that is, in general, best solved using a bio-inspired technique. We extend the results already published on this topic and show that Regression Random Forest can be a better alternative to solve the problem. A comparison of the method with previously published results is included. In order to implement the algorithm in a way that is as efficient as possible, a massively parallel implementation using a Graphics Processing Unit was used and is also described.

1. Introduction

Since the operation of a power plant is an expensive endeavor, there is a great demand for reliable and accurate ways and methods to make the operation as efficient as possible, be it in terms of fuel consumption or making decisions about the way to utilize the power plant's capabilities to fit the current demand for electricity. One of the ways to help achieve such goals is to predict the plant's output power based on some parameters that can be measured during its operation. This is often done using some optimization technique. Among the successful ones, Swarm Optimization was shown to provide interesting results [1]. Optimizing a power plant's operation, however, is not limited to swarms – [2] presents the Bee Colony Optimization as a way to approach the problem, a fuzzy model trained by genetic algorithms was used in [3] while [4] elaborates on multi-objective optimization with respect to a plant's operation economy.

Since the output power is usually a real number (or integer at best), the task of predicting the power based on input variables clearly suggest a regression problem. In this paper, we elaborate on the results already provided by Tüfekci in [5] and extend them with the suggestion to use the Regression Random Forest (RRF) algorithm. The method used by Tüfekci that is most similar to RRF is the WEKA implementation of the REPTree class, possibly extended by bagging. However, REPTree creates a number of trees and then selects the one that fits the training data best. RRF provides an option to use multiple trees in unison, eventually improving the results. This is elaborated in more detail later in this paper. The parallel implementation of RRF also described in this paper brings advantages over its classical implementations in terms of computation speed.

In this research, real-world data from a Combined Cycle Power Plant (CCPP), which consists of gas turbines, steam turbines and heat recovery system generators, was used. The parameters on which the power output is predicted are selected based on what individual parts of the plant are sensitive to. While gas turbines are sensitive to ambient variables such as temperature, pressure and relative humidity, steam turbines are highly influenced by the exhaust steam pressure, also called the vacuum. These are the four parameters that are used as input values in this study. The predicted variable is the electric power P_E generated by the plant. A more detailed description of the power plant can be found in [5].

2. Decision trees

To provide a briefer and easier-to-understand explanation of how decision trees work, we provide a description of the classification version of the tree, with transition to Random Forest and, finally, its regression version. However, the modifications to create the regression version are not that complex and are well described in [6].

* Corresponding author.
E-mail addresses: jan.janousek@vsb.cz (J. Janoušek), pavel.dohnalek@vsb.cz (P. Dohnálek).

http://dx.doi.org/10.1016/j.swevo.2015.07.004
2210-6502/© 2015 Elsevier B.V. All rights reserved.

The decision tree is a classification method that has been known for some time [7]. It is a binary tree where each node except for leaves consists of decision rules that determine which branch to go through next. The leaves of the tree contain a specific class. When estimating the class of an input vector, the tree is traversed from its root to the leaves.

The decision rules in individual nodes can be seen as functions. These functions are often referred to as *split functions* or *weak learners* [8]. They are defined as

$$f(x,\theta): \chi \times \tau \rightarrow 0,1, \tag{1}$$

where x is a given vector from the input set χ and θ are the parameters of a test function from the set τ. This function performs mapping to the *false* and *true* values in order to determine which branch of the tree should be selected for continuation.

There is a large number of split functions that can be used [9]. The most common and least computationally demanding is the *axis-aligned hyperplane*. This split function is determined by an index of the feature (according to which the split is made) and a limit value (a threshold). Mores split functions are frequently used, such as the *general oriented hyperplane*, *conic learner* and other both linear and non-linear functions [10–12]. It is important to mind that using a more complex function can provide better results, but may be more computationally demanding.

When building a decision tree, the goal is to find the best parameters for the split function. It is an optimization problem in which we seek to find the parameters that minimize the classification error. This error is proportional to the probability of selecting a particular class during random selection from the set of vectors that were assigned to the same group based on the split function.

As a metric for describing the quality of a split using a specific split function, *information gain* or *Gini impurity* is often used. In this work, we selected *information gain* which is based on the concept of entropy from information theory. Information gain is given by

$$G(S) = -\sum_{i=0}^{C} p(c_i)\log(p(c_i)), \tag{2}$$

where C is the number of classes and $p(c_i)$ is the probability of selecting a vector of class i from the vector set S. The optimization goal is to maximize the improvement obtained by the split based on the following formula:

$$I = G(S) - \sum_{i \subset \{L,R\}} \frac{|S_i|}{|S|} G(S). \tag{3}$$

In a similar way, we can define the entropy for continuous output values. To construct a decision tree, it is necessary to define the conditions that determine the tree's growth termination point. This condition can be, for example, the fact that after the last split there are only vectors belonging to a single class in the current branch. This condition is not flawless, because it can lead to a phenomenon known as overfitting, a situation in which the tree loses its ability to generalize. A better condition may be, for example, a limit for the information gain that was obtained with the last split. The final class of a leaf is then determined by the maximum likelihood of selecting the class from the set of vectors belonging to that branch. At the same time we are able to read from the leaf the probability that the class was determined correctly. An example of a decision tree, namely the Classification and Regression Tree (CART) in its classification mode, is pictured in Fig. 1.

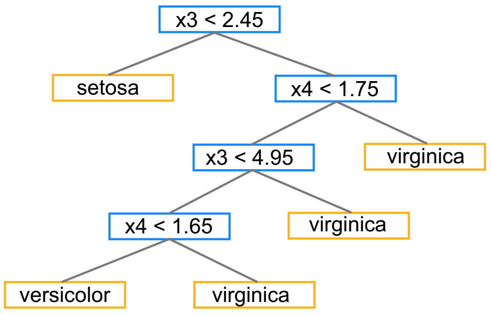

Fig. 1. An example of a decision tree, in this case a Classification and Regression Tree. This is a well-known example from MATLAB, portraying classification procedure of flowers, based on 4 input parameters (x1 through x4). The tree is strictly binary, the condition in a blue node (a decision rule), if met, allows to transition into the left subtree. The right subtree is visited if the condition is not met. The orange leaf nodes represent the final classification. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

2.1. Random Forest

Although the decision tree has been known for a considerable time, it has experienced a renaissance in recent years. This is due to the discovery that an ensemble of slightly different trees can provide better generalization for previously unseen inputs [13]. Because of this, the algorithm is more robust and can produce results more accurate than other models.

One of the algorithms that utilize this is the Random Forest. This algorithm uses randomness when building each of the many trees it merges into a single ensemble technique. Each new tree is created from a subset of the input vectors. This process is called *bagging*. From a set of all inputs, a certain percentage of vectors is selected and these vectors are then used to build a new tree. The selections are performed independently of each other, leading to some of them being present in the training set for more than one individual tree while others may not be included in any training set at all. Bagging helps to reduce the variance of the data and avoid tree overfitting.

Bagging is not the only part where randomness is utilized. Random Forest performs random optimization of the split functions. This means that the split function parameters are selected randomly. If the axis-aligned hyperplane is used, these parameters are the indices of the features that will be used for the split and threshold. This approach ensures that the newly grown trees will differ and provide margin-maximization [14]. Margin-maximization, in turn, provides better generalization and higher accuracy. The same principle is used, for example, in the Support Vector Machine [15,16].

The classification result of the entire forest is given by voting that can be expressed as

$$C_{rf} = voting\{C(x_i)\}_1^N, \tag{4}$$

where C_{rf} is the classification answer of the entire forest, x_i is the i-th observation being classified and N is the number of trees in the forest.

We can also explain how a combination of existing models reduces the total classification or regression error. There are three basic types of errors incurred during the classification task.

1. *Noise* – quantifies the output deviation from the optimal answer.
2. *Bias* – the average error of the model with regards to the optimal answer.
3. *Variance* – error that describes the error rate of the model during training on various learning sets.

The total model error is given as the sum of the above mentioned errors. As describing the solution to minimize this