Expert Systems with Applications 38 (2011) 14290–14300

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Decision tree induction using a fast splitting attribute selection for large datasets

A. Franco-Arcega [a,*], J.A. Carrasco-Ochoa [a], G. Sánchez-Díaz [b], J.Fco. Martínez-Trinidad [a]

[a] Computer Science Department, National Institute of Astrophysics, Optics and Electronics, Luis Enrique Erro #1, Santa Maria Tonantzintla, Puebla, C.P. 72840, Mexico
[b] Computational Science and Technology Department, University of Guadalajara, CUValles, Carretera Guadalajara-Ameca Km. 45.5, Ameca, Jalisco, C.P. 46600, Mexico

ARTICLE INFO

Keywords:
Decision trees
Large datasets
Gain-ratio criterion

ABSTRACT

Several algorithms have been proposed in the literature for building decision trees (DT) for large datasets, however almost all of them have memory restrictions because they need to keep in main memory the whole training set, or a big amount of it, and such algorithms that do not have memory restrictions, because they choose a subset of the training set, need extra time for doing this selection or have parameters that could be very difficult to determine. In this paper, we introduce a new algorithm that builds decision trees using a fast splitting attribute selection (DTFS) for large datasets. The proposed algorithm builds a DT without storing the whole training set in main memory and having only one parameter but being very stable regarding to it. Experimental results on both real and synthetic datasets show that our algorithm is faster than three of the most recent algorithms for building decision trees for large datasets, getting a competitive accuracy.

1. Introduction

Classification is an important task in data mining (Tan, Steinbach, & Kumar, 2006). Currently, there are many classification problems where large training datasets are available, therefore there is a big interest for developing classifiers that allow handling this kind of datasets in a reasonable time.

Decision trees (Quinlan, 1986, 1993) are commonly used for solving classification problems in Machine Learning and Pattern Recognition. A DT is formed by internal nodes, leaves, and edges, and it can be induced from a training set of instances, each one represented by a tuple of attribute values and a class label. Internal nodes have a splitting attribute and each node has one or more children (edges). Each one of these children has associated a value for the splitting attribute and these values determine the path to be followed during a tree traversal. Each leaf has associated a class label. In order to classify a new instance, the tree is traversed from the root to a leaf, when the new instance arrives to a leaf it is classified according to the class label associated to that leaf.

Several algorithms have been developed for building DTs from large datasets (Alsabti, Ranka, & Singh, 1998; Domingos & Hulten, 2000; Gehrke, Ramakrishnan, & Ganti, 1998, 2000, 1999; Mehta, Agrawal, & Rissanen, 1996; Shafer, Agrawal, & Mehta, 1996; Yang, Wang, Yang, & Chang, 2008; Yoon, Alsabti, & Ranka, 1999). However, almost all of them have spatial restrictions, because they have to keep the whole training set in main memory and some other use a representation of the attributes that requires more space than the whole training set. On the other hand, in those algorithms without spatial restrictions, the construction of the DT is based only on a small subset, but for obtaining this subset additional time is required, which could be too expensive for large training sets; or the algorithms uses several parameters, which could be very difficult to determine.

Having these drawbacks identified, this work introduces a new algorithm for building DTs that solves these problems. Our algorithm (DTFS) follows two main ideas for building DTs, it uses a fast splitting attribute selection for expanding nodes (deleting the instances stored in the expanded node after its expansion) and processes all the instances of the training set in an incremental way, therefore it is not necessary to store the whole training set in main memory.

In the literature some new techniques to select splitting attributes have been proposed (Berzal, Cubero, Marn, & Snchez, 2004; Chandra & Paul Varghese, 2009; Ouyang, Patel, & Sethi, 2009), however these techniques are not proposed for handling large datasets, because some of them have to evaluate a lot of candidate splits for choosing the best attribute, other use discretization methods to deal with numerical attributes, and some other use expensive techniques to expand nodes. On the other hand, several algorithms for building DTs in an incremental way have been proposed, such as ID5R (Utgoff, 1989), PT2 (Utgoff & Brodley, 1990), ITI (Utgoff, 1994), StreamTree (Jin & Agrawal, 2003) and UFFT (Gama & Medas, 2005), however these algorithms cannot handle large datasets either, because they need to keep the whole training set in main memory for building the DT.

In this paper, we propose an algorithm that processes the training instances one by one, thus each training instance traverses the

* Corresponding author. Tel.: +52 222 2663100x8311; fax: +52 222 2472580.
E-mail address: anifranco6@inaoep.mx (A. Franco-Arcega).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.05.087

DT until a leaf is reached, where the training instance will be stored. In our algorithm, when a leaf has stored a predefined number of instances (a parameter of the algorithm), it will be expanded choosing a splitting attribute, using the instances in the leaf, and creating an edge for each class of instances in the leaf. After expanding a leaf, the instances stored in that leaf are deleted. Experimental results over several large datasets show that our algorithm is faster than three of the most recent algorithms for building DTs for large datasets, obtaining a competitive accuracy.

The rest of the paper is organized as follows. Section 2 gives an overview of the works related to DT induction for large datasets. Section 3 introduces the DTFS algorithm, which allows building DTs for large datasets. Section 4 provides experimental results and a comparison against other algorithms for DT induction for large datasets, on both real and synthetic datasets. Finally, Section 5 gives our conclusions and some directions for future work.

2. Related work

In this section, several algorithms that have been proposed to build DTs for large datasets are described.

Mehta et al. (1996) presented SLIQ (Supervised Learning In Quest), an algorithm for building DTs for large datasets. This algorithm uses a list structure for each attribute, these lists are used in order to avoid storing the whole training set in main memory, by storing them in disk. However, SLIQ uses an extra list that must be stored in main memory, this list contains the class of each instance and the number of the node where this instance is stored in the tree. This could be a problem for large datasets, because the size of this list depends on the number of instances in the training set. The process that SLIQ follows to build a DT is similar to C4.5, but the difference is that SLIQ uses the lists for splitting attribute selection, therefore, the lists must be read from disk each time a node is going to be expanded.

Shafer et al. (1996) presented an improvement of SLIQ, called SPRINT (scalable parallelizable induction of decision trees). The difference with respect to SLIQ lies in how SPRINT represents the lists for each attribute. SPRINT adds a column to each list for storing the class of each instance, hence SPRINT does not need to store in main memory any whole list. However, since SPRINT has to read from disk all the lists for expanding each node, just like SLIQ, the runtime is too large if the training set has a lot of instances.

Alsabti et al. (1998) proposed CLOUDS (Classification for Large or OUt-of-core DataSets), an algorithm that uses, as SLIQ and SPRINT, lists for representing the information of the attributes in the training set. However, these lists are simplified representing the numerical attributes by intervals. This modification substantially reduces the time required for choosing the attributes that will represent the internal nodes of the DT, because it is not needed to check all the values for each attribute. A drawback of SPRINT and CLOUDS is that for storing the lists they require at least the double of the space needed for storing the original training set.

Gehrke et al. (1998), Gehrke, Ramakrishnan, and Ganti (2000) introduced the Rainforest algorithm. It follows the idea of using lists for representing the attributes of a training set, but this algorithm only stores all the different values for each attribute. In this way, Rainforest tries to reduce the size of the lists, thus the list size will not be the number of training instances but the number of different values. However, these lists must be stored in main memory, therefore if the attributes have a lot of different values in the training set, the available space could be not enough. Besides, in order to generate the lists, in each level of the DT, Rainforest has to read the whole training set twice and write it once, which is very expensive for large datasets. Nguyen and Tae-Choong (2007) presents an improvement of Rainforest, the difference is that this improvement adds to the lists, used by Rainforest, the position of each instance in the training set, in order to use them in the expansion of each node, in this way this algorithm does not have to read twice and write once the whole training set in each level of the tree. This algorithm only scans once the whole training set, however if an attribute has too many values, its lists may not fit in main memory.

Gehrke, Ganti, Ramakrishnan, and Loh (1999) developed an incremental algorithm for building DTs, called BOAT (Bootstrapped Optimistic Algorithm for Tree construction). This algorithm avoids to store the whole training set in main memory by using only an instance subset as training for building the DT, however obtaining this subset requires additional time for building the DT, which could be expensive for large datasets. Starting from this subset, BOAT applies a bootstrapping technique for generating multiple DTs, using a traditional main memory DT induction algorithm (for example C4.5, CART, etc.). The constructed DTs are combined, and finally, BOAT refines the combined DT using the whole training set.

Yoon et al. (1999) proposed another incremental algorithm for building DTs, called ICE (Incrementally Classifying Ever-growing large datasets). This algorithm divides the training set in subsets, called epochs, and processes them separately, therefore ICE does not need to store the whole training set in main memory. ICE builds a DT for each epoch using a traditional main memory DT induction algorithm (as C4.5, CART, etc.) and from each DT, ICE obtains a subset of instances applying a sampling technique. Then ICE joins the subsets, obtained from each epoch, for building the final DT. A DT T_i is built from each subset D_i (epoch i) of the training set, and using a sampling technique a set S_i of samples is extracted from T_i. The union of S_i and the previous sets of samples are stored in U_i. Then the new set of samples $U_i = U_{i-1} \cup S_i$ is preserved for building the DT in the next epoch. For a training set divided in k epochs, ICE joins $S_1, S_2, \ldots, S_k$, the subsets of instances extracted from $T_1, T_2, \ldots, T_k$, and builds the final DT C_k with the last U_k, the algorithm preserves the subset U_k and the DT C_k. If a new epoch D_{k+1} must be processed, ICE builds T_{k+1} from D_{k+1}, extracts S_{k+1} from T_{k+1} and uses $U_{k+1} = U_k \cup S_{k+1}$ for building the new DT, C_{k+1}. A drawback of ICE is that when the algorithm processes large training sets, it spends a lot of time for obtaining the subset of instances for building the final DT.

Domingos and Hulten (2000) introduced an incremental algorithm called VFDT (very fast decision trees). This work proposed the Hoeffding trees, which can be learned in constant time per instance, and they are similar to the trees built by traditional main memory DT induction algorithms (for example C4.5, CART, etc.). For building the DT, VFDT needs the training instances in a random order, if this is not the case, they should be randomized in a preprocessing step. VFDT starts with a tree produced by a conventional DT induction algorithm, this tree is built from a small subset of instances, then VFDT processes each instance of the training set traversing the DT and updating the statistics needed to compute the information gain of each attribute in the leaf in which the instance arrives. VFDT uses a user-defined parameter n_{min}, which indicates the minimum number of instances that must be stored in a leaf before checking if the node has enough information to be expanded. When n instances have arrived to a leaf, VFDT obtains the information gain of each attribute using the statistics stored in the leaf, chooses the two attributes with highest information gain, $G(X_a)$ and $G(X_b)$, and obtains the Hoeffding bound ε using Eq. (1).

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{1}$$

In Eq. (1) R is $log(c)$ (c is the number of classes) and δ is a user-defined parameter, where $1 - \delta$ indicates the probability of