Knowledge-Based Systems 92 (2016) 9–22

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Windowing improvements towards more comprehensible models

Pedro Santoro Perez [a], Sérgio Ricardo Nozawa [b,1], Alessandra Alaniz Macedo [a], José Augusto Baranauskas [a,*]

[a] Department of Computer Science and Mathematics (FFCLRP), University of Sao Paulo (USP), Av. Bandeirantes 3900, Ribeirão Preto, SP 14040-901, Brazil
[b] Dow AgroSciences (Seeds, Traits & Oils), Av. Antonio Diederichsen 400, Ribeirão Preto, SP 14020-250, Brazil

ARTICLE INFO

Article history:
Received 24 July 2015
Revised 11 September 2015
Accepted 4 October 2015
Available online 22 October 2015

Keywords:
Windowing
Decision tree metrics
High dimensional data

ABSTRACT

The induction of decision tree searches for relevant characteristics in the data which would allow it to precisely model a certain concept, but it also worries about the comprehensibility of the generated model, helping human specialists to discover new knowledge, something very important in the medical and biological areas. On the other hand, such inducers present some instability. The main problem handled here refers to the behavior of those inducers when it comes to high-dimensional data, more specifically to gene expression data: irrelevant attributes may harm the learning process and many models with similar performance may be generated. In order to treat those problems, we have explored and revised windowing: pruning of the trees generated during intermediary steps of the algorithm; the use of the estimated error instead of the training error; the use of the error weighted according to the size of the current window; and the use of the classification confidence as the window update criterion. The results show that the proposed algorithm outperform the classical one, especially considering measures of complexity and comprehensibility of the induced models.

1. Introduction

One of the greatest challenges for machine learning (ML) is to build precise models out of high-dimensional data, like gene expression data. Furthermore, in biology and medicine, the comprehensibility and interestingness of the model is also important for the specialists to trust it and have insights about the phenomena being studied, which might lead to creating new knowledge [1,2]. In this paper, we have studied the performance of decision trees (DTs) built from high dimensional data like gene expression, focusing on not only measures like accuracy, but also on those related to the comprehensibility of the classifiers.

On the other hand, given the fact that ML algorithms, including DTs, have been designed for a broad spectrum of the human knowledge, they are based on flexible and powerful representations. This flexibility may, in some cases, present the disadvantage of making inducers more susceptible to the training data. DTs, for example, are known to produce very different models with small changes in the data, which may have the opposite effect: the specialists get confused and stop trusting the models [3,4]. This paper has also evaluated how to improve the stability of DTs.

Specifically, in gene expression data, the number of genes (attributes) is typically much greater than the number of tissue samples (instances), and only a small subset of the genes is relevant to the task at hand [5,6]. The great advantage of DTs over other learning paradigms, such as function-based and statistical paradigms, is that they possess an embedded process of attribute selection, allowing them to use only those attributes which were considered to be relevant and informative [7]. As a result, the final DT models are smaller, syntactically simpler and more comprehensible. Considering stability again, in general, DTs built from gene expression data tend to include very few attributes in the model; however, given the huge number of attributes, many trees with similar accuracy can be built, each of which with a different subset of attributes [8].

Gene expression profiles are obtained *via* Molecular Biology techniques like Microarray [9] (based on intensity measures of DNA hybridization) and SAGE [10] (based on tag counting). This kind of data helps in the discovery of diagnostic and prognostic methods, diseases treatment, drug development [11], *etc.* Many research works have concentrated efforts on applying ML algorithms on gene expression analysis [1,2,12–19].

In order to deal with the problems described above, we have used windowing [20], a technique whose idea is to find a subsample of a dataset that provides enough information for an inducer to train a classifier. It has results similar to those achieved by training a model from the entire dataset, reducing the complexity of the learning problem [21]. This way, windowing can be seen as a subsampling

* Corresponding author. Tel.: +55 16 33154439; fax: +55 16 33150407.
E-mail addresses: pedrosperez@usp.br (P.S. Perez), srnozawa@gmail.com (S.R. Nozawa), ale.alaniz@usp.br (A.A. Macedo), augusto@usp.br (J.A. Baranauskas).
[1] Phone: +55 16 36025650; fax: +55 16 36025696

http://dx.doi.org/10.1016/j.knosys.2015.10.011
0950-7051/

technique [22], but, unlike other subsampling techniques (*e.g.*, bootstrapping), windowing tend to provide more class balanced and informative samples. The technique was first proposed by [23] in the context of decision trees as a way to deal with memory restrictions imposed by computers in the late 1970's to some relatively large datasets. The work of [20] argues windowing is interesting for two reasons: i) in some cases, especially those free of noise, it may make the time taken to build the model shorter (*e.g.*, when the dataset is very large and a model that perfectly classifies all training instances is achieved in the first iterations). For most cases, however, windowing makes that time longer; ii) windowing may produce more accurate classifiers, since it explores the solution space a little further. Memory is still an issue, given the existence of huge databases on medical and biological domains.

The study performed by [24] states that windowing contributes to better threshold choices for continuous attributes in decision tree induction. In [25], windowing is seen as a way to make unstable learning algorithms more stable. Some other studies have explored windowing in the context of rule induction [26], as they have noted the technique produces better results with that kind of inducer than with decision trees, since rules are learned independently of each other and are less susceptible to changes in the class distribution. In spite of the experience Quinlan had with windowing, those studies also state that the technique is not good to be used with decision trees, especially in noisy domains [27]. Possibly because of that, very little has been researched towards improving the combination between windowing and decision trees, although there still are some work on it [28]. Our research group believes there still are some aspects of windowing that could be explored further and has proposed some changes in the original algorithm, which will be discussed in the next sections. These changes are focused on improving the performance of decision trees built from gene expression data (which are typically noisy).

Our implementation of windowing can be applied to any learning algorithm that works with classification. Our group has been focusing on symbolic classifiers, *i.e.*, those which can be written as a set of rules. In this context, one advantage of windowing over other meta-inducers is that its use with symbolic classifiers still provides symbolic classifiers. Instead of just concerning about accuracy, we have used comprehensibility-related measures to assess the models. Such measures are well established when it comes to rule induction, but not when it comes to DTs. We have proposed some of those measures but specifically applied to DTs.

The remaining sections are organized as follows: Section 2 gives details about the original version of windowing; Section 3 describes the alterations proposed in windowing and how the experiments were conducted; Section 4 presents the measures used to assess the results; Section 5 presents the experimental setup; results and discussion can be found in Section 6; conclusions are presented in Section 7.

2. Original windowing

The version of windowing used here is that of C4.5 [20], an algorithm of top-down induction of decision trees. Starting from the root of the tree, the algorithm performs a greedy search for attributes to be used in tests in the internal nodes. Each internal node performs a test on only one attribute and has two or more branches, each of which representing a test outcome. The leaf nodes contain class labels. To label a new example: from the root node, tests are performed and, according to their outcomes, the example goes down the tree until it reaches a leaf node, receiving its label. The decision tree inducer has a Java implementation in Weka [29] called J48.

This section describes windowing as it is implemented in C4.5 Release 8 [20]. Algorithm 1 shows the pseudo-code of windowing, where N represents the number of instances in the training set and

Algorithm 1 Original windowing.

Require: Instances: a set of N labeled instances $\{(\vec{x}_i, y_i), i = 1, 2, \ldots, N\}$
W: the initial window size, default value $W \leftarrow \max\{0.2N, 2\sqrt{N}\}$
I_0: the tentative increment size, default value $I_0 \leftarrow \max\{0.2W, 1\}$
Ensure: bestClassifier: the best classifier found

```
1:  N ← |Instances|
2:  window ← sample(W, Instances)
3:  bestClassifierErrors ← N + 1
4:  repeat
5:    h ← induceClassifier(window)
6:    windowErrors ← Σ_{x_i ∈ window} ‖ h(x_i) ≠ y_i ‖
7:    testErrors ← Σ_{x_i ∉ window} ‖ h(x_i) ≠ y_i ‖
8:    totalErrors ← windowErrors + testErrors
9:    if (totalErrors < bestClassifierErrors) then
10:     bestClassifier ← h
11:     bestClassifierErrors ← totalErrors
12:   end if
13:   I ← max{min{testErrors, I_0}, testErrors/2}
14:   I ← min{I, N − |Instances \ window|}
15:   increment ← get I first misclassified test instances
16:   window ← window ∪ increment
17: until testErrors = 0
18: bestClassifier ← prune(bestClassifier)
19: return bestClassifier
```

$\vec{x}_i$ and y_i ($i = 1, \ldots, N$) represent a vector containing the attribute values and the class label for instance i, respectively. The $\|E\|$ operator returns 1, if E is true, or 0, otherwise.

Before the learning process begins, a subset of the training set is chosen, forming the initial window (Line 2), from which a model is induced (Line 5); the model is then used to predict the class of all training examples inside (windowErrors) and outside the window (testErrors), which might produce some misclassifications (Lines 6–7); if the errors found are less than the errors of the best classifier so far (initially, $N + 1$), the current classifier is kept as the best one (Lines 9–12); if there were errors outside the window, the window is updated and used to train another classifier; the resulting model is then tested again, and the process is repeated until no misclassifications occur outside the window.

The initial window (Line 2) is not actually sampled randomly. First the training set is shuffled; then the algorithm tries to build a window as uniform as possible, *i.e.*, the class distribution gets to be as balanced as possible. Considering c as the actual number of all class values of a dataset and $\mathcal{E} = W/c$ as the expected number of instances for each class value in the initial window W, for any given class value, if the number of instances labeled with it is at least equal to $\mathcal{E}$, then it will be represented by $\mathcal{E}$ instances in the initial window; otherwise, all instances representing that class value will be added to the window. This often leads to better results, especially in cases of unbalanced classes [20]; it also contributes to better threshold choices for continuous attributes [24]. In spite of being as uniform as possible, the class distribution and the examples chosen to be in the initial window are subject to a random sampling, which means that one may have different initial windows, leading to different final classifiers. As it can be seen in Algorithm 1, at least half of the misclassified examples outside the window is added to it at each iteration (Lines 13–16), provided that there are enough examples. This is done to make the model converge faster. Lines 15–16 take I misclassified examples from outside the window and adds them to it.

The process can be repeated more than once. Each repetition is called a *trial* and starts with a different initial window, which often generates a different final classifier. By default, C4.5 uses 10 trials. The best tree classifier from all trials is returned as the final output.