Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 2168478, 5 pages
http://dx.doi.org/10.1155/2016/2168478

Research Article

A Searching Method of Candidate Segmentation Point in SPRINT Classification

Zhihao Wang,[1] Junfang Wang,[1] Yonghua Huo,[1] Yanjun Tuo,[1] and Yang Yang[2]

[1]*Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang, China*
[2]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*

Correspondence should be addressed to Zhihao Wang; cetc540016@sina.com

Received 5 April 2016; Revised 5 August 2016; Accepted 1 September 2016

Academic Editor: Bin-Da Liu

SPRINT algorithm is a classical algorithm for building a decision tree that is a widely used method of data classification. However, the SPRINT algorithm has high computational cost in the calculation of attribute segmentation. In this paper, an improved SPRINT algorithm is proposed, which searches better candidate segmentation point for the discrete and continuous attributes. The experiment results demonstrate that the proposed algorithm can reduce the computation cost and improve the efficiency of the algorithm by improving the segmentation of continuous attributes and discrete attributes.

1. Introduction

In recent years, with the rapid development of economy and the continuous improvement of the level of computer technology, a large number of databases are used in business management, scientific research, and engineering development. In the face of massive storage data, how to find valuable information is a very difficult task. Data mining is to help people to extract valuable information from large, incomplete, random fuzzy data. Classification is a very important section in data mining. The purpose of classification is to construct a function or a model by which data can be classified into one of the given categories. The classification model can achieve the goal of forecasting data [1, 2]. The prediction model is derived from historical data records to represent the trend of the given data, so that it can be used to forecast future data.

The ID3 algorithm is a significant algorithm for building a decision tree [3, 4]. The information gain is used in this algorithm to select node's attributes in a decision tree. But ID3 has the shortcoming of inclining when choosing attributes in the large scale values. The improved method C4.5 is proposed based on the ID3 algorithm [5, 6], and the C4.5 method uses the information gain rate instead of the information gain to select attributes of the decision tree, which improves the efficiency of decision trees. Then many improved algorithms based on the ID3 algorithm have been proposed, including SLIQ, SPRINT, and other algorithms. The SLIQ [7] algorithm can handle classification of large datasets. The SPRINT algorithm [8–10] based on SLIQ can be unrestricted by memory and its processing speed is considerable.

The SPRINT algorithm has many advantages. This algorithm is unrestricted by memory, and it is a kind of scalable and parallel method of building decision trees. But there are also some shortcomings. For example, finding the best segmentation point of discrete attributes needs a large amount of calculation, and the partition of continuous attributes is unreasonable.

Based on these issues, this paper proposes a new method of searching for the best segmentation point. For the segmentation of discrete attributes, the new method reduces time complexity by avoiding unnecessary computation. For the segmentation of continuous attributes, we can achieve the goal of reducing the depth of decision trees and improving the classification efficiency of decision trees through discretization of continuous attributes.

2. Related Works

Decision tree is one of the most widely used classification models in machine learning applications. Its goal is to extract

knowledge from large scale datasets and represent them in a graphically intuitive way.

The paper [1] presents the Importance Aided Decision Tree (IADT), which takes feature importance as an additional domain knowledge for enhancing the performance of learners. Decision tree algorithm finds the most important attributes in each node. Therefore, the mechanism of importance of features in the paper is a relevant domain knowledge for the decision tree algorithm. For automatically designing decision tree, Barros et al. [2] propose a hyperheuristic evolutionary decision tree algorithm tailored to a specific type of classification dataset. The algorithm evolves design components of top-down decision tree induction algorithms.

The key of ID3 algorithm is considering information gain as the reference value for testing attributes, which leads to lower classification accuracy [3]. So the authors in [4] proposed a new scheme for solving the shortcoming of ID3. The paper uses the improved information gain based on dependency degree of condition attributes as a heuristic when it selects the best segmentation attribute.

Ersoy et al. [5] proposed an improved C4.5 classification algorithm with the hypothesis generation process. The algorithm adopts k-best Multi-Hypothesis Tracker (MHT) to reduce the number of generated hypothesis especially in high clutter scenarios.

In order to solve the security problems of intrusion detection system (IDS), attack scenarios and patterns should be analyzed and categorized. The enhanced C4.5 [6] is a combination of tree classifiers for solving security risks in the intrusion detection system. The mechanism uses a multiple level hybrid classifier which relies on labeled training data and mixed data. Thus, the IDS system based on C4.5 mechanism can be trained with unlabeled data and is capable of detecting previous attacks.

SLIQ decision tree solves the problem of sharp decision boundaries which are hardly found in classification. Thus the paper [7] proposes a fuzzy supervised learning in Quest decision tree. The authors construct a fuzzy decision boundary instead of a crisp decision boundary. In order to avoid incomprehensible induction rules in a large and deep decision tree, fuzzy SLIQ constructs a fuzzy binary decision tree, which has significant reduction in tree size.

SPRINT decision tree algorithm can predict the quality level of system modules, which is good for software testing [8]. The paper presents an improved SPRINT algorithm to calibrate classification trees. It provides a unique tree-pruning technique based on the minimum description length (MDL) principle. Based on this, SPRINT tree-based software quality classification mechanisms are used to predict whether a software module is fault-prone or not fault-prone.

3. SPRINT Algorithm

3.1. Description of SPRINT Algorithm. The SPRINT algorithm has no limit to the number of input records and its processing speed is considerable. This algorithm creates a list of attributes and a corresponding statistics table for each attribute of the sample data in the initialization phase. Elements in the list of attributes are known as attribute records, which consisted of labels, attribute values, and classes. Statistics tables are used to describe the class distribution of a property, and the C above and C below two lines, respectively, describe the class distribution of processed samples and untreated samples.

Steps of the original SPRINT algorithm are as follows:

Maketree (node s) {

If (node s meets the termination conditions) {

Put node s into the queue, labeled as a root node;

Return;

}

For (for each attribute A) {

Update histogram in real time;

Calculate and evaluate the index of segmentation for each candidate segmentation points, and find the best segmentation point;

Find out the best segmentation for node s from the best segmentation for each attribute. Based on it make two part S_1, S_2;

Maketree (S_1);

Maketree (S_2);

}

}

The termination condition of the algorithm has three kinds of cases. (1) No attribute can be used as testing attribute. (2) If all the training samples in the decision tree belong to the same class, the node is used as a leaf node and labeled by this class. (3) The number of training samples is less than the user-defined threshold.

3.2. Segmentation of Attributes. The traditional SPRINT algorithm uses *Gini* index [5] to search for the best segmentation attribute, which provides the minimum *Gini* index representing the largest information gain.

For a dataset D containing N classes, *Gini* is defined as

$$\text{Gini}(D) = 1 - \sum_{j=1}^{N} p_j * p_j. \tag{1}$$

p_j is the frequency of class J in D. If a partition divides the dataset D into two subsets D_1 and D_2, $|D_1|$ and $|D_2|$ represent the number of records in subsets D_1 and D_2, respectively. After the segmentation, the *Gini* value is

$$\text{Gini}(S) = \frac{|D_1|}{|D|}\text{Gini}(D_1) + \frac{|D_2|}{|D|}\text{Gini}(D_2). \tag{2}$$

A segmentation of attribute values providing the least *Gini* value is chosen as the best segmentation [9].

For discrete attributes and continuous attributes, the SPRINT algorithm uses different processing methods.

In order to find discrete attribute segmentation point [7], we assume that the number of a certain attribute's values

is n, which should be divided into two parts. All attribute values are considered as possible partition, and then the corresponding *Gini* value is obtained. There are 2^n kinds of possible partitioning ways in total. We need to calculate the *Gini* value for each partitioning way using exhaustive method and then can obtain the best segmentation.

For the solution of finding the continuous attribute's partitioning point, the split can only occur between two values. First the values of the continuous attribute should be sorted and the candidate segmentation points are intermediate points between two values.

After a scan of sorted values, the statistics table should be updated when a record is read. The statistics table contains all the information needed to calculate the *Gini* index. Then we should calculate the *Gini* index to find the segmentation point with the minimum *Gini* value.

Although the traditional method can find the best segmentation point, it is necessary to traverse all of the segmentation in discrete attributes [8], which makes this algorithm have high time complexity. For the segmentation of continuous attributes, dividing them into two consecutive parts in most cases can not reflect the distribution of attribute values.

4. Improved SPRINT

4.1. Segmentation of Discrete Attribute. Taking credit risk of bank as an example, the data record is shown in Table 1.

Values of a discrete attribute with m kinds of classes are divided into two sets, and then there are 2^m types of partitions, which mean that the Gini index values should be calculated for 2^m times. In Table 1, there are four kinds of classes, student, worker, clerk, and retiree, so 2^4 kinds of partitions should be considered. Taking into account commutative law of addition in formula (2), Gini_{sp} value remains unchanged when exchanging attribute values in sets of D_1 and D_2. For example, $D_1 = \{\text{student}, \text{worker}\}$ and $D_2 = \{\text{clerk}, \text{retiree}\}$; in this case, $\text{Gini}_{sp} = 0.375$. When attribute values in D_1 and D_2 are exchanged, that is, $D_1 = \{\text{clerk}, \text{retiree}\}$ and $D_2 = \{\text{students}, \text{workers}\}$, $\text{Gini}_{sp} = 0.375$.

According to this property, the times of calculating Gini index can be reduced for segmentation of discrete attributes in the SPRINT algorithm. In order to reduce the time complexity of SPRINT algorithm, this paper proposes an improved discrete attribute partition algorithm.

D is a collection of discrete attribute values, and the number of values in D is M. Now the attribute value set D is divided into two sets D_1 and D_2. Select some values from D and put them into D_1. The number of selected values is i. The initial value of i is 1 with one-step growth until $i < M/2$. Values are identical in D_1 and D_2 in the case of $i > M/2$. When M is odd, it is impossible that i is equal to $M/2$. If M is an even number and i is equal to $M/2$, there are $C_M^{M/2}$ kinds of combinations of attribute values in D_1, and 1/2 of attribute combinations are the same as D_2. So after selecting values for D_1, we need search for the same collection in D_2. If there is the same collection in D_2, the collection in D_1 will be deleted. And when the number of D_1 is more than $1/2 * C_M^{M/2}$, the search is stopped.

Table 1: Credit risk records of bank.

Number	Profession	Risk
1	Student	High
2	Worker	High
3	Clerk	High
4	Clerk	Low
5	Worker	Low
6	Retiree	High
7	Retiree	High

At the same time when all values in a subset belong to the same class, this subset can be a leaf node that does not need to be partitioned. So we can ignore two cases of {all values} and {empty}. In summary, this paper firstly proposed a new algorithm to reduce calculation of candidate segmentation points for discrete attributes. There are M kinds of different values in a discrete attribute, and the improved algorithm on discrete attributes is as follows.

Step 1. Initialize a class partition table (including four fields: number, first collection, second collection, and *Gini* value), and set the counter $i = 1, n = 1$.

Step 2. If $i < M/2$, i values are placed in the first collection of the class partition table and the *Gini* index of this division is calculated and then carry on for the next time.

Step 3. Step 2 ends; $i = i + 1$; compare i with $M/2$. If $i < M/2$, then return to Step 2; if $i = M/2$, execute the next step; if $i > M/2$, skip to Step 6.

Step 4. Put i values in the first collection and the others into the second collection. Search for the values of the first collection in the list of the second collection. Find out if there is a second collection same as the first collection. If there is, this partition will be deleted; otherwise calculate the *Gini* index of this partition.

Step 5. $n = n+1$; compare n with $1/2 * C_M^{M/2}$; if $n \leq 1/2 * C_M^{M/2}$, skip to Step 4. If not, skip to the next step.

Step 6. Find out the minimum *Gini* value based on the optimized class partition table.

It can be seen that the improved algorithm eliminates repeated operations and unnecessary operations, which reduces computation greatly and reduces the time of creating a decision tree.

4.2. Segmentation of Continuous Attribute. Candidate segmentation points are middle points of two continuous values for segmentation of continuous attributes in SPRINT algorithm, and the attribute values are divided into two parts by a middle point. For example, there are two values V_1 and V_2, and their middle point $V = (V_1 + V_2)/2$ is a candidate segmentation point for a continuous attribute. Values (V_i) of this continuous attribute which are less than V belong to a

collection, and the values greater than V belong to the other collection. However, in many cases this segmentation method is not conducive to the classification of the target attribute.

This algorithm includes three steps: sorting, classifying, and random combination of continuous attributes. And classifying is a new idea that is not included in the SPRINT algorithm. A continuous attribute A has values $\{a_1, a_2, a_3, \ldots, a_i\}$. The target attribute E has positive and negative examples of $\{P, N\}$. According to the target attribute values continuous attribute values are classified. For target attribute value P, collection 1 is $\{a_{p1}, a_{p2}, \ldots, a_{pi}\}$ and $\{a_{p1}, a_{p2}, \ldots, a_{pi}\} \in \{a_1, a_2, a_3, \ldots, a_i\}$; the corresponding record's target attribute value of a_{pi} is P. Similar to N, collection 2 is $\{a_{n1}, a_{n2}, \ldots, a_{ni}\}$; $\{a_{p1}, a_{p2}, \ldots, a_{pi}\} \cup \{a_{n1}, a_{n2}, \ldots, a_{ni}\} = \{a_1, a_2, a_3, \ldots, a_i\}$. And the values in collection 1 and collection 2 are sorted in ascending or descending order. The following processing is performed on collection 1 and collection 2.

Step 1. Calculate the difference between two neighboring values $c_i = a_{p(i+1)} - a_{pi}$ ($c_i = a_{n(i+1)} - a_{ni}$).

Step 2. Sort the series of c_i in descending order. Find the top x values in series c_i, and the corresponding a_{pi} and $a_{p(i+1)}$ (a_{ni} and $a_{n(i+1)}$) are candidate segmentation points.

Step 3. There are $2 * x$ candidate segmentation points in collection 1 (collection 2). $4 * x$ candidate segmentation points are found in total.

Step 4. Sort $4 * x$ candidate segmentation points in ascending or descending order, $\{v_1, v_2, v_3, v_4, \ldots, v_{(4*x)}\}$.

Step 5. $r_i = v_{(i+1)} - v_i$; find the minimum value in series r_i. The corresponding v_i and $v_{(i+1)}$ are deleted. Add $(v_i + v_{(i+1)})/2$ in series r_i.

Step 6. Repeat Step 5 until the number of series of r_i is x.

Step 7. Divide all values into $x+1$ blocks using x segmentation points. The values of continuous attribute have been divided into $x + 1$ blocks through the above steps, and consider these $x + 1$ blocks as $x + 1$ discrete attribute values; then the segmentation method of discrete attribute values is used to process these blocks.

Step 8. Initialize a class partition table (including four fields: number, first collection, second collection, and *Gini* value), and set the counter $i = 1$, $m = 1$.

Step 9. If $i < (x + 1)/2$, i blocks are placed randomly in the first collection of the class partition table; the *Gini* index of this division is calculated and then carry on for the next time.

Step 10. If splitting process ended, then set $i = i + 1$ and compare i with $(x + 1)/2$. If $i < (x + 1)/2$, then return to the previous step; if $i = (x + 1)/2$, execute the next step. If $i > (x + 1)/2$, then jump to Step 13.

Step 11. Put i blocks in the first collection and the others into the second collection. Search for the blocks in the list of the second collection and find out the values as same as the first collection. If there is, this partition will be deleted; otherwise calculate the *Gini* index of this partition.

Step 12. $m = m + 1$; compare m with $1/2 * C_{x+1}^{(x+1)/2}$; if $m \leq 1/2 * C_{x+1}^{(x+1)/2}$, return to the previous step. If not, proceed to the next step.

Step 13. Find out the minimum *Gini* value based on the optimized class partition table.

Steps 8–13 are the same as the improved algorithm on discrete attributes.

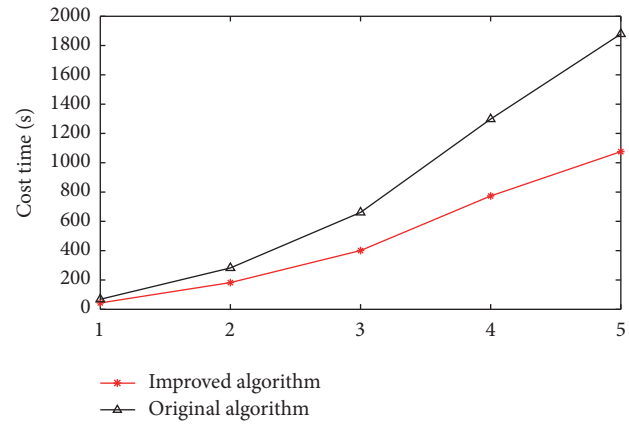

Figure 1: Comparison of the SPRINT algorithm and the improved SPRINT algorithm.

5. Experiment and Simulation

This experiment uses the dataset of Function [11] as experimental samples. Attributes of the dataset include age, salary, vocation, level, and other attributes. There are discrete attributes, for example, vocation, and continuous attributes, for example, age in the dataset. The VC++ 6.0 is the experiment platform for this experiment. Comparison of the original SPRINT algorithm [9] and the improved SPRINT algorithm is shown in Table 2.

Visualization of data on Table 2 is shown as in Figure 1.

The quantities of data in the five sets are increasing, so the costing time is also growing. As shown in Figure 1, the improved SPRINT algorithm greatly reduces the time to generate decision trees. At the same time, the classified accuracy of the decision tree generated by the improved SPRINT algorithm is also tested.

The comparison results of classification accuracy are shown in Table 3.

As shown in Table 3, the improved SPRINT algorithm almost has the same or slightly better classification accuracy ratios as the original algorithm. With the increasing scale of dataset, the classification accuracy ratios have accordingly decreased. The decision tree becomes larger with the increase of the amount of data, which may result in the decreasing in

Table 2: Comparison of the SPRINT algorithm and the improved SPRINT algorithm.

Dataset (10^6)	Costing time of original algorithm (s)	Costing time of improved algorithm (s)	$\frac{\text{Costing time of improved algorithm}}{\text{Costing time of original algorithm}} \times 100\%$
1	68	43	63.24%
2	282	182	64.54%
3	661	401	60.67%
4	1298	773	59.55%
5	1879	1076	57.26%

Table 3: Classification accuracy of original algorithm and improved algorithm.

Dataset (10^6)	Accuracy of original algorithm (%)	Accuracy of improved algorithm (%)
1	95.8%	98.61%
2	88.7%	89.8%
3	83.8%	83.7%
4	80.2%	80.5%
5	77.4%	77.5%

accuracy. Controlling the size of the decision tree needs to be further researched.

6. Conclusion

In summary, the improved SPRINT algorithm improves the calculation for searching the best segmentation by searching better candidate segmentation point for the discrete and continuous attributes, which reduces the unnecessary operations, increases the speed of generating decision trees, and reduces the time cost greatly.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this article.

Acknowledgments

This work was supported by Open Subject Funds of Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory (ITD-U15002/KX152600011).

References

[1] M. R. A. Iqbal, S. Rahman, S. I. Nabil, and I. U. A. Chowdhury, "Knowledge based decision tree construction with feature importance domain knowledge," in *Proceedings of the 7th International Conference on Electrical and Computer Engineering (ICECE '12)*, pp. 659–662, Dhaka, Bangladesh, December 2012.

[2] R. C. Barros, M. P. Basgalupp, A. A. Freitas, and A. C. P. L. F. De Carvalho, "Evolutionary design of decision-tree algorithms tailored to microarray gene expression data sets," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 6, pp. 873–892, 2014.

[3] I. Kamwa, S. R. Samantaray, and G. Joós, "On the accuracy versus transparency trade-off of data-mining models for fast-response PMU-based catastrophe predictors," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 152–161, 2012.

[4] H. He, T. M. McGinnity, S. Coleman, and B. Gardiner, "Linguistic decision making for robot route learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 203–215, 2014.

[5] Y. Ersoy, M. Efe, and B. Nakiboglu, "Enhancement of multiple hypothesis tracking algorithm with C4.5 algorithm," in *Proceedings of the 20th Signal Processing and Communications Applications Conference (SIU '12)*, pp. 1–4, April 2012.

[6] L. P. Rajeswari and A. Kannan, "An Intrusion Detection System based on multiple level hybrid classifier using enhanced C4.5," in *Proceedings of the International Conference on Signal Processing Communications and Networking (ICSCN '08)*, pp. 75–79, January 2008.

[7] B. Chandra and P. P. Varghese, "Fuzzy SLIQ decision tree algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 5, pp. 1294–1301, 2008.

[8] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski, "Decision trees for mining data streams based on the McDiarmid's bound," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1272–1279, 2013.

[9] M. Guijarro, R. Fuentes-Fernández, P. J. Herrera, Á. Ribeiro, and G. Pajares, "New unsupervised hybrid classifier based on the fuzzy integral: applied to natural textured images," *IET Computer Vision*, vol. 7, no. 4, pp. 272–278, 2013.

[10] M. L. Othman, I. Aris, S. M. Abdullah, M. L. Ali, and M. R. Othman, "Knowledge discovery in distance relay event report: a comparative data-mining strategy of rough set theory with decision tree," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2264–2287, 2010.

[11] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.