A Novel Tree Cluster and Classification Approach Based on Least Closed Tree

Xin Guo

School of Software and Service Outsourcing, Jishou University, Zhangjiajie, Hunan 427000, China

jianghai079@126.com

Keywords: Data mining; Tree mining; Closed tree pattern; Tree cluster; Tree classification

Abstract. The extensive application of tree model has made tree mining become a hot field in data mining research. As an important branch of tree mining, tree cluster and tree classification plays a fundamental analysis role in many areas. In this paper, a tree cluster and classification algorithm was proposed based on least closed tree, which effectively solved problems in large amount of data in practical application. The basic method is bringing forward least closed tree as the candidate cluster and classification feature, using dynamic threshold by similarity cluster to make tree cluster operation be more quick and accurate, meanwhile the concept of tree classification rule grade proposed is used in tree classification algorithm, so that the unknown tree structure could be predicted promptly. Experimental results show that the method has higher speed and efficiency than that of other similar ones especially when large number of tree nodes.

Introduction

As the tree model can accurately denote the key feature of science and engineer data, therefore the research of tree mining has attracted more and more people's attention. Data in many fields could be abstracted as trees, such as the XML structure can be described as labeled ordered tree [1, 2], biology, computer networks, WWW data could also be abstracted as trees, which is used for further analysis.

In recent years, researchers has made plenty of results on tree mining [3, 4, 5], as an important branch, tree cluster and tree classification has extensive application in areas like biology and internet analysis etc. Many algorithms based on tree cluster and classification has been brought forward. Papers [6, 7, 8] respectively put forward tree cluster algorithms on XML document tree. In these algorithms, XML documents are abstracted as labeled ordered trees and tree mining method is adopted. The problem of these algorithms is that when dealing with small amount of data (few tree nodes), they could run smoothly, however, a relatively large database with numbers of tree nodes could significantly decrease the efficiency, and even can not run, paper[8] proposed XML tree similarity measure algorithm, the number of nodes has become a bottleneck for the algorithm. Regards to tree classification, paper [9] proposed a classification method based on frequent pattern, the main problem of this method is how to set an minimum support during frequent pattern mining, a high support will lead a small amount of result and affect the performance of classifier, yet a low support will make the algorithm become low efficiency so that the mining task could not be completed. Paper [10] put out a method based on the graph kernel function classification and by using support vector machine to carry out classification, whereas this method is of high classification performance only for compound structures, and it is not feasible in other applications.

In this paper, a common cluster and classification method based on least closed tree for large scale database was proposed. Algorithm generates feature set by using least closed tree at first, and then process tree cluster in accordance with the proposed method of similarity measure, which could get all class set and corresponding category, the last step is to predict tree category of the new tree structure by taking use of tree classification method. The main contribution of this paper is as follows:

1) Proposed the least closed tree as the candidate feature of tree clustering and classification, makes tree mining algorithm could use a low support and consequently generates more candidate

feature set which will take an important role during tree cluster and classification, furthermore, a candidate feature set composed by least closed tree could not only reserve the useful information for tree cluster and classification, but also its quantity is less than the candidate feature composed by frequent subtree.

2) Brought forward the least closed tree mining algorithm, and effectively solved the problem of couldn't take cluster and classification operation when large data amount in practical application.

3) Put forward tree similarity measure algorithm so that the bottleneck problem in paper [10] could be solved, meanwhile a tree cluster method based on dynamic threshold was proposed.

4) Put out tree classification rule grade and tree classification method, which could accurately predict the category of a new tree.

5) Experimental results show that the method provided in this paper is better than that of similar ones in cluster and classification operation on biological database like Ribonuclease P [12] etc.

Backgrounds

Definition 1. Labeled Ordered Tree[1]. A rooted labeled ordered tree $T = (V, E)$ is a directed, acyclic, connected graph with V as the set of vertices and $E = \{(x, y) \mid x, y \in V\}$ as the set of edges. l : $V \to L$ is a labeling function mapping vertices to a set of labels $L = \{l_0, l_1, \ldots, l_n\}$. l_i is the label of node. Order means all sub nodes of each node in the tree form a fraternal relationship from left to right.

Like pre-visiting binary tree, pre-visiting a tree means visiting root first and then each subtree from left to right. During this process, the order of each node receiving its visit forms pre-node number, short named as node number. During pre-visiting, the order of each node receiving its visit forms a string of node number that is called the string show of the tree.

Definition 2. Tree Database. Let TDB denote a database of ordered tree, the single tree of TDB is represented as $(TID, String)$, the TID is the tree ID in the database, and the String is string of tree.

Definition 3. Matching and Appearance [11, 15]. If a tree T and a data tree D exist a mapping $\phi : V_T \to V_D$, and the mapping satisfies the following propositions:

1) . The mapping ϕ between tree T and data tree D is a one-to-one correspondence.

2) . $L_T(v) = L_D(\phi(v))$.

3) . $(v_1, v_2) \in E_T iff (\phi(v_1), \phi(v_2)) \in P_D$.

4) . $preorder(v_1) < preorder(v_2)$, denotes the ordered number of node v after the preorder traversal of all tree nodes.

Then calling the ϕ is a matching function from tree T to data tree D, i.e., tree T appears in data tree D, the T is a subtree of data tree D. If a tree has k nodes, then calling it is a k-subtree, and the empty tree matches with any tree.

Definition 4. Support. Let an ordered tree database TDB be $TDB = \{T_i \mid i = 1, 2, \cdots, n\}$, the absolute support of tree T in TDB is defined the number of trees containing tree T in TDB, denoted as $abs\_\sup(T, TDB) = |\{T_i \mid T_i \in TDB, T \subseteq T_i\}|$. The relative support of tree T in TDB is defined the percent of trees including tree T in TDB, denoted as $rel\_\sup(T, TDB) = abs\_\sup(T, TDB) / |TDB|$.

Definition 5. Frequent Subtree. Given an ordered tree database $TDB = \{T_i \mid i = 1, 2, \cdots, n\}$ and minimum relative support $\min\_\sup$, if a subtree T satisfies $rel\_\sup(T, TDB) \geq \min\_\sup$, then calling the subtree T is a frequent subtree in TDB, the set of all frequent subtrees in TDB is denoted as $F(TDB) = \{T \mid rel\_\sup(T, TDB) \geq \min\_\sup\}$.

Definition 6. Closed Tree. Let an ordered tree database TDB be $TDB = \{T_i \mid i = 1, 2, \cdots, n\}$, and let the T be a frequent subtree, if there doesn't exist proper super-trees in the ordered tree database TDB,

then calling tree T is a closed tree in TDB, the set of all frequent closed subtrees in the tree database TDB is denoted as $CF(TDB)$.

The problem of tree cluster may be described in detail as follows: given an ordered tree database $TDB = \{T_i | i = 1, 2, \cdots, n\}$, researching some methods to gain r non-intersecting clusters $\{\beta_1, \beta_2, \cdots, \beta_r\}$. In this paper, we may choose some representative trees in each cluster, calling these trees is a tree class, thus we are able to acquire a set of tree class, denoted as $\{t_1^*, t_2^*, \cdots, t_r^*\}$.

Definition 7. Least Closed Tree. Let the $T = \{t_1, t_2, \cdots, t_v\}$ be a closed tree set, $t \in T$, for $\forall t' \in T$, if the t' is not a subtree of tree t, then calling tree t is a least closed tree. The least closed tree t^* of which the number of nodes is smallest in all least closed trees is regarded as the tree class.

The problem of tree classification may be described in detail as follows: inputting r clusters $\{(\beta_i, t_i^*) | i = 1, 2, \cdots, r\}$ and a tree T with unknown classification, where the β_i is a tree set, the t_i^* is a tree class, outputting some cluster that the tree T belongs to.

Algorithm

Least Closed Tree Algorithm. In this paper, we concern the mining of the least closed tree. Mining the least closed tree has the following several advantages compared with mining the entire frequent subtree: firstly, though the amount of the frequent closed trees is less than frequent subtrees. In the practical application, we can not only get all frequent subtrees but only can not get the frequent closed trees, so we only get the least closed tree in this paper. For example, in the Ribonuclease database application, the total files of the database are nearly 2MB. When the minimum support is 1%, it will have at least 8GB frequent subtrees files, and will have nearly 200MB frequent closed trees files, but a large numbers of the data have high similarities in there data. For example, some subtrees only have different edges, or only have different nodes. We only need mine a feature set, and it can speed up the mining process, and speed up the (classification) and the clustering process at the same time. Secondly, the smaller the minimum support is ,the more accurate the clustering and classification information will be .If we use the smaller minimum support to mine all the frequent subtrees and all the closed tree, for instance ,we set the minimum support as 1%, and we can get a lot of subtrees, and sometimes we can not complete the task . So if we only mine the least closed trees, we can use the smaller minimum support to get a lot of information of the subtrees which are useful for classification and clustering. The most important is that mining closed trees can not lose any useful information of the classification and clustering, because we can revive all the frequent trees from the closed trees. In the practical application, we only need a feature set or a sample set instead of a complete subtrees set, and the experts are only interested in the feature set or the sample set.

ITMSV [17] is an ordered tree mining algorithm. The algorithm uses the right extended method to generate the candidate subtrees, and generate the subtree based on the connection strategy. It can generate all the candidate subtrees to utilize the first connection strategy, and generate the corresponding vector of the subtrees based on the second connection. When we get the candidate subtrees , We can get the information of the support at the same time because the subtree vectors include all the information of the subtrees in the database, for example, the number of the subtrees, and we need not scan the database as well as can speed up the Operating efficiency .

In this paper, we propose a least closed tree mining algorithm based on the

ITMSV algorithm defines a pruning process, and deletes all non-closed tree structures based on the nature of the closed tree. The process checks all the subtrees if there exist a same support with the subtree, and we only need find the hash key based on the node information because all the frequent subtrees information have been saved in the hash tables. This algorithm also defines a random selector to choose the frequent closed trees randomly, and defines two precision $\beta = [m, n]$ (m, n are the non-negative decimal) and $\lambda > 1$ to Control the number of the generated frequent trees.

For example, when we set β as $[0.1,0.2]$ and set λ as 10000, we retain this subtree if the data which is generated by the random selector is in the rage of β, otherwise we delete it, and we judge if the total number of the subtree surpasses λ. If there are 10000 feature trees, the algorithm is stopped. Finally, this algorithm extracts the entire least closed tree according to the closed trees set.

Firstly, the algorithm scans the database to get the frequent 1-subtree, and then it uses all the frequent 1-subtrees to Completed a hash table, and puts the hash table into a linear table structure. We can get the frequent 2-subtrees according to mongering the two keys in the hash table based on the Candidate generation strategy.

The algorithm checks all the frequent 2-subtrees if there exits a same support with the subtree calling pruning procedures, if there exists, we delete the 2-subtrees, and otherwise we can choose a non-negative decimal according to the system time calling the random selector. If the non-negative decimal is in the range of β,we should reserve the subtree, otherwise we should delete the subtree. We must judge the total number of the generated tree if it is in the scope of λ, and if it surpasses, the algorithm is stopped. Finally, we build a new hash table which contains all the remaining frequent 2-subtrees, and put it into a table structure at the same time. The keys of the hash table are the keys of the tree, and the keys of the hash table contain the information of the subtree vector, then we can get the frequent 3-subtrees. We can call the pruning procedures and the random selection procedures in the same way, and put the hash table which includes the remaining frequent 3-subtrees into the table structure. We can get all the frequent closed tree structures, and in the end, the algorithm can generate all the least closed trees sets.

The algorithm of the LeastClosedTreeMine is the following:

Algorithm 1 Procedure *LeastClosedTreeMine*
Input: the database of tree: D , support , α, β, γ.
Output: all least closed trees.

```
1:  LeastClosedTreeMine (TDB,minusp) :
2:  F1={frequent 1-subtrees}; Hash ←F1;
3:  Line[1] ←Hash and Node=1 and TotalNum=0;
4:  While(Line[Node]!=null) do
5:     Node=Node+1; new_hash=null;
6:     For all [T] in Hash do
7:        For each element (x, i) ∈[T] do
8:           For each element (y,j) ∈[T] do
9:           R={(x,i) ⊕(y,j)};
10:          If Node<α then
11:             If all subtrees of R if frequent then
12:             Delete R;continue;
13:          L(R)={ L(X,i) ⊕L(Y,j)};
14:          If support(R)>minusp AND Check(R)
     AND Rand()∈β then
15:             TotalNum++;New_hash←R;
16:             If TotalNum>γ then return;
17:          End IF
18:    End for all;
19:    If new_hash!=null then
20:       Line[Node] ←new_hash;
21: End while
22: End LeastClosedTreeMine
```

Tree Similarity. Similarity measurement methods include the comparison of the characteristics and the calculation of the conversion costs and so on. We can also combine the several methods. To measure the similarity. The similarity of the two trees structure can be measured by the ratio which the same information that both include has in the total information. Given the two tree structures, the formula for calculating the similarity can be defined as:

$$sim(T_a, T_b) = \frac{|t(T_a) \cap t(T_b)|}{|t(T_a) \cup t(T_b)|} \quad (1)$$

In this paper, we proposed the characteristics comparison method to measure the similarity based on the calculation of the XML document similarity[16]. We should not only calculate the semantic similarity, but we also calculate the structural similarity considering the structural characteristics of the tree. When we get the semantic similarity and the structural similarity, we can add the weights to the two aspects respectively to get the total similarity ,and in the practical application, we can find that structural characteristics of trees are often more important than semantic characteristics. For example, Biologists tend to give more focus on the structure of a particular molecular structure, and excellent structural site allows users to browse the Web quickly and easily, so we set the two weights which have a larger margin in the similarity comparison.

Definition 8. Expansion Vector [8]. We set na as the tree nodes. $Ex(na)=(na,na^1,na^2,\cdots,na^m)$ is the expansion vector of na, and $na^i=(1,2,\cdots,m)$ are the synonyms, compound words or the acronym forms of na.

Definition 9. We set t_1, t_2 as the tree nodes logo, and we define the scores of similarity between t_1, t_2 as the following [8]:

6) If t_1 matches t_2 completely

5) If t_1 matches certain elements besides t_2 in the $Ex(t_2)$ completely or t_2 matches certain elements besides t_1 in the $Ex(t_1)$ completely.

4) If certain elements besides t_1 in the $Ex(t_1)$ matches certain elements besides t_2 in the $Ex(t_2)$ completely.

3) If t_1 matches t_2 partly.

2) If t_1 matches certain elements besides t_2 in the $Ex(t_2)$ partly or t_2 matches certain elements besides t_1 in the $Ex(t_1)$ partly.

1) If certain elements besides t_1 in the $Ex(t_1)$ matches certain elements besides t_2 in the $Ex(t_2)$ partly.

0) there exits no matching

The formula which is used as the similarity measurement between t_1 and t_2 for any two tree structure t_1, t_2 is as the following:

$$Sim(t_1,t_2)=\lambda_1 SemSim(t_1,t_2)+\lambda_2 StruSim(t_1,t_2)$$

In the formula, $SemSim(t_1,t_2)$ is the semantic similarity, and $SemSim(t_1,t_2)$ is the structural similarity. λ_1,λ_2 are the weights of the semantic similarity and the structural similarity respectively. In the actual calculation, we set λ_1 as 0.2, and set λ_2 as 0.8.

When we calculate the semantic sim*ilarity, we firstly calculate one vector value of the following two according to the number of the tree nodes:*

$$t_1=\left(\left\langle Ex\left(l_1^1\right),score_1^1\right\rangle,\cdots,\left\langle Ex\left(l_m^1\right),score_m^1\right\rangle\right)$$

$$t_2=\left(\left\langle Ex\left(l_1^2\right),score_1^2\right\rangle,\cdots,\left\langle Ex\left(l_n^2\right),score_n^2\right\rangle\right)$$

We only need calculate one vector value of the two as the value of the semantic similarity, because that structural characteristics of trees are often more important than semantic characteristics. If the number of nodes of t_1 is smaller than t_2 , we compute the vector of t_1 , and otherwise, we compute the vector of t_2 .The $Ex\left(l_i^j\right)$ is the expansion vector of the node logo of the tree, and $score_i^j$ is the score value of the similarity of the node l_i^j. We compare $score_i^j$ and the node which is the most similar to it in the other tree, and give the score of the similarity for all the nodes computed once, and compute it as the following:

$$SemSim(t_1,t_2)=\sum_{i=1}^{m} score_i^j \Big/ 6k \tag{2}$$

In the formula (2), k is the number of the nodes of the trees which have the smaller nodes, and the sum of the $score_i^j$ is the score value of the similarity of the corresponding tree .This algorithm can save the time and speed up the efficiency when there are a lot of nodes.

We firstly mark the nodes of the tree based on the depth-first method, and give the similar nodes the same number. When we do this, each path of the node trees can be expressed by a digital sequence. We can find the similar path of the frequent sequence of the two subtrees, and then we can compute it as the following:

$$StruSim(t_1,t_2)=\frac{1}{N+1}\left[\left(\sum_{t=1}^{N}\frac{1}{L(R_t)}\times V(R_t)\right)+MR\right]$$

$$V(\begin{array}{l}R_t \text{ is leaf}\\ R_t \text{ is not leaf}\end{array}\begin{array}{l}\cdot\\ \mid\end{array}\begin{array}{l}1\\ \dfrac{1}{N(C_t)\sum_{e\in C}\frac{1}{L(e)}V(e)}\end{array}$$

$$\begin{array}{l}R_t \text{ is similarity}\\ R_t \text{ is not similarity}\end{array}\begin{array}{l}\cdot\\ \cdot\end{array}R_t)=\begin{cases}1\\0\end{cases}$$

Tree Cluster Algorithm. The algorithm in this paper firstly calls the least closed tree mining algorithm to gain a least closed tree set, then achieves cluster analysis by calculating the similarity between trees. At last, all trees are classified according to the similarity. In order to cluster exactly, we will choose threshold ε dynamically. We randomly select a subtree t from the least closed tree set, and calculate the similarity between tree t and others in the least closed tree set. Then we rank all similar values by descending to form a non-increasing curve, after that, we find the inflexion of the non-increasing curve by computing the second derivative. At last, the inflexion closest to the coordinates' origin is regarded as the threshold ε of similarities, all subtrees whose similar thresholds exceed the ε form a candidate set, if the number of elements of the candidate set exceeds the specified-user density threshold $d>1$, then these subtrees come into being a classification. Afterwards, we delete these subtrees from the least closed tree set, and adopt similar method to get other classifications for the rest of the least closed tree set. If the number of elements of the candidate set is less than the specified-user density, we will renewably choose a subtree, and then newly calculate according to the above-mentioned method.

For all acquired classifications, we compute their tree labels. All tree labels constitute a set, denoted as $\{t_1^*,t_2^*,\cdots,t_r^*\}$, where the t_i^* denotes the tree label of classification i. The computing method of the t_i^* is the following: the tree label of a classification is a least closed tree t^* of which the number of nodes is least.

The algorithm of the *TreeCluster* is the following:

Algorithm 2 Procedure *TreeCluster*
Input: least closed trees: T , density: D.
Output: all class C and Tc.

```
1:  TreeCluster (T,d) :
2:  While(T!=null) do
3:     t=rand();
4:     For all temp in T do
5:        Sim(t,temp);
6:     End for all;
7:     threshold=Sort();
8:     S={all|Sim(all,t)> threshold};
9:     If |S|>=threshold then
10:       C←S; T=T- S;
11:    End IF
12: End while
13: For all c in C do
14:    Tc←gen(c);
15: End TreeCluster
```

Tree Classification Algorithm. After the algorithm in this paper finishes running and gains r classifications $\{\beta_1, \beta_2, \cdots, \beta_r\}$, the algorithm in this article classifies a new tree t further. By all appearances, each closed tree p is corresponding to a class rule $p \to t_i^*$ in a certain classification β_i, where the t_i^* represents a classification, and the β_i is corresponding with the tree label. We may directly make use of these classification rules to classify a new instance; however, there exist a problem in this way. In order to reserve these closed trees which are important for classifying prediction, the support usually is specified a small value, the algorithm can only gain the least closed tree set, but the number of least closed trees is still large. If all closed trees are executed operations of tree matching, then it is referred too many computations. In practice, the number of tree nodes often is so large that it restricts practical application for classification methods. Consequently, we must choose some decisive classification rules for classifying in all classification rules. Before introducing class methods, we first define a rank of trees' classification rules.

Definition 10. Classification rules' rank of trees. For any two rules t_1 and t_2, if one of the following conditions is established, then the rank of rule t_1 is higher than that of rule t_2, denoted as $t_1 \succ t_2$.

1) The support of rule t_1 is larger than that of rule t_2.

2) The support of rule t_1 is the same as rule t_2, but the number of its nodes is larger than rule t_2

3) The support of rule t_1 is the same as rule t_2, and their nodes are also same, however, rule t_1 is outputted earlier than rule t_2.

According to the classification rules' rank of trees, all classification rules may be arrayed linearity order from large to small; we choose n rules from large to small. For the sake of classifying exactly, we respectively calculate the average of similar grades between the t and each classification in r classifications. The function is the following:

$$Equal(t, \beta_i) = \sum_{k=1}^{n} Sim(t, t_k) \Big/ n$$

Where the β_i represents the classification i, $t_k \in \beta_i$, the parameter n denotes the number of rules whose ranks are higher in classification β_i. If the n is larger than the number of all rules, then the n is namely the number of all rules. According to all averages of similar grades, it is feasible to predict the classification of the t, i.e., the classification which the maximum average of similar grade corresponds to is as a new tree's category.

The algorithm of the *TreeClassification* is the following:

Algorithm 3 Procedure *TreeClassification*
Input: class: C, trees: t , number: n.
Output: category.

```
1:  TreeClassification (C,t,n) :
2:  Score[];
3:  Sort(C);
4:  For all c in C do
5:      Score←–Equal(t,c);
6:  End for all;
7:  max=Max(Score);
8:  End TreeClassification
```

Experimental

We implement all the algorithms of this paper using C++, and for these algorithms, we have carried out a large number of experiments with different parameters running on experimental environments (Pentium(R) 4 CPU 2.80GHz, Memory 1GB, Hardware 120G, OS Red Hat Linux6.0). We adopt thread to calculate the time of the algorithms.

We test the algorithms with artificial database, and to truly prove the effectiveness of them, we use a common artificial database generator, which is from document [10], using 8 parameters to adjust the distribution of result data. The 8 parameters are S (the size of the label set), p (the probability of one node can generate child node), L (the number of the base tree), I (base tree height), C (the fan-out of each node of base tree), N (the size of TDB), H (the max height of each tree in TDB), F (the fan-out of each node of TDB). The base tree and the tree height in TDB are both follow the Gaussian distribution with expectation to be $I(H)$ and standard deviation to be 1. The default parameter of result data is S100 P0.5 L10 I4 C3 N100000 H8 F6 and the default support is 1%.

We compare the ATFC algorithm in document [7] and the Xproj algorithm in document [8] with the Tree Cluster algorithm in this paper with different parameters, and the experimental results has been shown in Figure 1. And then we compared the Tree Classification algorithm with the XRules algorithm in document [10] to all of the result class sets. We conduct experiments in accordance with the increment of complexity (fan-out, height) of tree for the class set which contains 10 unclassified trees, and the experimental results has been shown in Figure 2.

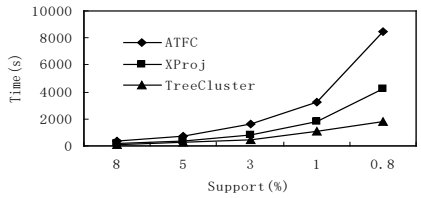

Figure 1. run time vs threshold of support

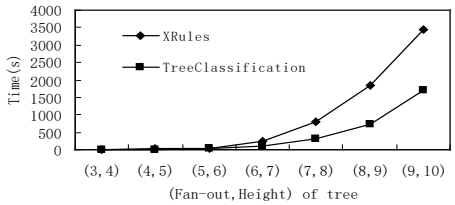

Figure 2. runtime vs complexity of tree

We cluster and classify the Ribonuclease P data in biology. Due to too large for the amount of data, we use a higher support here than in artificial database, and so, using clustering algorithm with different parameters, we have the experimental results shown in Figure 3 and using classification algorithm with the increment of complexity (fan-out, height) of tree, we have the experimental results shown in Figure 4.

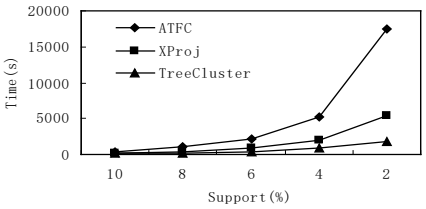

Figure 3. run time vs threshold of support

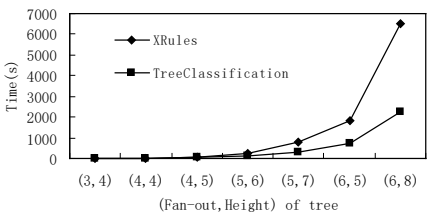

Figure 4. runtime vs complexity of tree

We analyses web log using classification algorithm and clustering algorithm of this paper. We download the web log from Adaptive Web Sites (https://www.cs.washington.edu/research/adaptive/) from which we select a portion from Sept.20, 1999 to Oct.4, 1999. Then from the huge amount of data, we get that about 500,000 logs to visit cs.washington.edu, and convert it to a tree. From it we get the set of class relationship shown in Table 1.

Table 1. the set of class relationship

LABEL1	LABEL2
research	projects
education	courses
people	faculty
…	…

Conclusions

A cluster and classification method aimed at large scale database was introduced in this paper, which provided a common solution to many practical applications. Proposed a cluster and classification method based on least closed tree, so that a low support could be retained and more candidate feature set that of important role in cluster and classification will be generated. Least closed tree mining algorithm referred in this paper effectively solved the problem of couldn't undergo the operation of cluster and classification on large data amount in practical applications. Adopted dynamic threshold degree selection as well as similarity cluster method, so that could separate the tree structures with different similarity, thus it can carry out tree cluster operation quickly and accurately. Introduced the conception of tree classification rule grade, and put this conception into tree classification operation, accordingly it could predict unknown tree structure accurately. Numbers of experimental results show that the method in this paper is effective and feasible under the condition of many tree nodes and large amount of data and it is obviously better than other methods in the application of Ribonuclease P. The tree cluster and classification method in this paper is base on labeled ordered tree, for future work, we will consider taking more features like labeled unordered tree or unlabeled tree into the analysis of cluster and classification, meanwhile expansion tree mining including graph mining [13, 14], graph cluster and classification could also employ this method, all of which are arranged in the next research plan.

References

[1] M.J.Zaki, "Efficiently mining frequent trees in a forest: Algorithms and Applications", In IEEE Transaction on Knowledge and Data Engineering, special issue on Mining Biological Data.Vol.17, No.8, pp 1021~1035, 2005.

[2] Y.Chi, Y.Yang, and R.R. Muntz, "Indexing and Mining Free Trees", Proc. Thind IEEE Int'1 Conf. Data Mining, 2003.

[3] Y.Chi, Y.Yang, and R.R. Muntz, "HybridTreeMiner:An Efficient Algorihtm for Mining Frequent Rooted Trees and Free Trees Using Canonical Forms", Proc.16th Int'1 Conf. Scientific and Statistical Database Management,2004.

[4] Y.Chi, Y.Yang,Y.Xia,and R.R.Muntz, "CMTreeMiner: Mining both closed and maximal frequent subtrees", In The Eighth Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04), May 2004.

[5] M.Hasan, V.Chaoji, S.Salem, j.Besson, and M.J.Zaki, "ORIGAMI: Mining Representative Orthogonal Graph Patterns", 7th IEEE International Conference on Data Mining, Omaha, NE, October 2007.

[6] T. Dalamagas, T. Cheng, K. Winkel, T. Sellis, "Clustering XML Documents Using Structural Summaries", Information Systems, Elsevier, January2005.

[7] C.C. Aggarwal, N. Ta, J. Wang, J. Feng, M.J.Zaki, “XProj: A Framework for Projected Structural Clustering of XML Documents”, SIGKDD’07.

[8] Wu Yangyang, Lei Qing, “A Method of Discovering Relation Information from XML DAata”, Journal of Software, 2008(6)：1422~1427.

[9] M Deshpande, M Duramochi, G Darypis, “Frequent substructure-based approaches for classifying chemical compounds”, IEEE Trans on Knowledge and Data Engineering, 2005, 17(8):1036~1050.

[10] T Horvath, T Gartner, S Wrobel, “Cyclic pattern kernels for predictive graph mining [C]”, KDD-2004, Seattle, USA, 2004.

[11] Zhao Chuanshen,etc. “Frequent Subtree Mining Based on Projected Branch.”, Journal of Computer Reseach and Development,2006 :456~462(in Chinese).

[12] J.W. Brown, “The Ribonuclease P Database”, Nucleic Acids Research, vol.27, no.1, pp.314-315,1999.

[13] V.Chaoji, M.A.Hasan, S.Salem, J.Besson, M.J.Zaki., “ORIGAMI: A Novel and Effective Approach for Mining Representative Orthogonal Graph Patterns”, Statistical Analysis and Data Mining, Vol. 1, Issue 2, pp. 67-84, (DOI: 10.1002/sam.10004) June 2008.

[14] Vineet Chaoji, Mohammad Al Hasan, Saeed Salem, Mohammed J. Zaki, “An integrated, generic approach to pattern mining: data mining template library”, Data Mining and Knowledge Discovery , Published Online: DOI:10.1007/s10618-008-0098-x, June 2008.

[15] Zhu Yongtai,etc, “ESPM—An algorithm to mine frequent subtrees”, Journal of Computer Reseach and Development, 2004(10): 1720~1726(in Chinese).

[16] Lee JW, Lee KH, Kim W, “Preparations for semantics-based XML mining”, In: Cercone N, Lin TY, Wu XD, eds. Proc. Of the 2001 IEEE Int’1 Conf. on Data Mining. Washington: IEEE Computer Society, 2001. 345~352.

[17] Yun Li, Xin Guo, Yun-Hao Yuan, “A Fast Algorithm of Mining Induced Subtrees”, Proc. of international conference on Information and Automation (ICIA 2008), 195~199.