International Journal of Engineering Research and Development
eISSN : 2278-067X, pISSN : 2278-800X, www.ijerd.com
Volume 2, Issue 2 (July 2012), PP. 24-27

A comparative study of various Multi Relational Decision Tree Learning algorithms

Prof. Saurabh Tandel

Abstract—The motive of this paper is to provide an introduction and insight into the family of Multi Relational Decision Tree Learning Algorithms by providing the comparative study. This paper will be a great help for those who wish to perceive their career in the area of Relational Decision Tree Induction. After the comparative study, it provides an improved version of the MRDTL[1](Multi Relational Decision Tree Learning) algorithm that proposes the use of the Tuple ID propagation instead of the complex data structures such as selection graph as are employed in the older generation of the algorithms.

Keywords—Decision Tree Learning, MRDTL, Information Gain, Naïve Bayes Predictor

I. INTRODUCTION

1.1 MRDTL Algorithm

```
TREE INDUCTION(D, S)
Input: Database D, selection graph S
Output: The root of the tree, T
R := optimal refinement(S)
if stopping criteria(S)
return leaf
else
Tleft := TREE INDUCTION(D,R(S))
Tright := TREE INDUCTION(D,R(S))
return node(Tleft, Tright,R)
```

Multi-relational decision tree learning algorithm[1] constructs decision tree whose nodes are Multi relational patterns i.e., selection graphs. The algorithm described as MRDTL is an extension of the logical decision tree induction[6] algorithm called TILDE. TILDE uses first order logic clauses to represent decisions (nodes) in the tree. The data are represented in first order logic rather than a collection of records in a relational database. MRDTL extends TILDE's approach to deal with records in relational databases. They use similar refinements operators, and the way the tree is inducted follows the same logic.

Essentially, MRDTL, like the propositional version of the decision tree algorithm adds decision nodes to the tree through a process of successive refinement until some termination criterion is met (e.g., correct classification of instances in the training set). Whenever some termination criterion is met, a leaf node with its corresponding class is introduced instead. The choice of decision node to be added at each step is guided by a suitable impurity measure (e.g., information gain) Thus, MRDTL starts with a single node at the root of the tree, which represents the set of all objects of interest in the relational database. The pseudocode for the algorithm is shown above. The function optimal_refinement considers every possible refinement that can be made to the current pattern G with respect to the database D and selects, in a greedy fashion, the optimal refinement (i.e., one that maximizes information gain). The possible set of refinements to be made at certain point during the process is governed by the current selection graph, the structure of the database D, and the multiplicity of the associations involved. G denotes the complement of the selection graph (i.e., it selects objects from the database that are not selected by G.

1.2 MRDTL-2(Improved MRDTL)

MRDTL[2] has two significant limitations from the standpoint of multi-relational data mining from large, real-world data sets:

--Slow running time

MRDTL (like other algorithms based on the multi relational data mining framework proposed by Knobbe uses selection graphs to query the relevant databases to obtain the statistics needed for constructing the classifier. The execution of queries encoded by such selection graphs is a major bottleneck in terms of the running time of the algorithm.

-- Inability to handle missing attribute values

In multi-relational databases encountered in many real-world applications of data mining, a significant fraction of the data has one or more missing attribute values. For example, in the case of gene localization task from KDD Cup 2001,

70% of CLASS, 50% of COMPLEX and 50% of MOTIF attribute values are missing. Leiva's implementation of MRDTL treats each missing value as a special value ("missing") and does not include any statistically well-founded techniques for dealing with missing values. Consequently, the accuracy of decision trees constructed using MRDTL is far from satisfactory on classification tasks in which missing attribute values are commonplace. For example, the accuracy of MRDTL on the gene localization task was approximately 50%.

These enhancements enable us to apply multi-relational decision tree learning algorithms to significantly more complex classification tasks involving larger data sets. MRDTL-2[5]

-- significantly outperforms MRDTL in terms of running time
-- yields results that are comparable to the best reported results obtained using multi-relational data mining algorithms
-- compares favorably with feature-based learners that are based on clever propositionalization methods

II. NEW PROPOSED ALGORITHM

2.1 Phase - I: Naïve Bayesian Classifier – Preprocessing

-- Useful for handling the missing values in the Data Sets by way of Probability Estimation[4].
-- Bayesian classifiers are statistical classifiers.

They can predict class membership probabilities, such as probability that a given tuple belongs to a particular class[8].

Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$. Suppose there are *m* classes $C_1, C_2, \ldots, C_m$. Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$. This can be derived from Bayes' theorem Since P(X) is constant for all classes, only

$$P(Ci/X) = P(X/Ci)P(Ci)$$

needs to be maximized

2.2 Phase - II: Tuple ID Propagation

In essence, tuple ID[7] propagation is a method for virtually joining non-target relations with the target one, and it is a simple but efficient method. It is much less costly than physical join in both time and space. Suppose the primary key of the target relation is an attribute of integers, which represents the ID of each target tuple. Consider the sample database shown below. Traditionally, the two relations will be physically joined. Also, they can be virtually joined by ID propagation, which is more efficient, as shown below.

Tuple ID propagation is a flexible and efficient method. IDs and their associated class labels can be easily propagated from one relation to another relation. By doing so, the next computing tasks can do with little redundant, and the required space is also small than the physical join.

Loan

loan-id	account-id	amount	duration	payment	class
1	124	1000	12	120	+
2	124	4000	12	350	+
3	108	10000	24	500	–
4	45	12000	36	400	–
5	45	2000	24	90	+

Account

account-id	frequency	date
124	monthly	960227
108	weekly	950923
45	monthly	941209
67	weekly	950101

Loan

loan-id	account-id	···	class
1	124		+
2	124		+
3	108		–
4	45		–
5	45		+

Account

account-id	frequency	date	IDs	class labels
124	monthly	960227	1, 2	2+, 0–
108	weekly	950923	3	0+, 1–
45	monthly	941209	4, 5	1+, 1–
67	weekly	950101	–	0+, 0–

2.3 The RDC algorithm

Given a relational database with one target relation, RDC uses the basic idea of MRDTL to build the decision tree. RDC starts with a single node at the root of the tree, which corresponds to the target relation Rt. It adds nodes to the tree through a process of successive refinement until some termination criterion is met. The choice of refinement to be added at each step is guided by information gain. It chooses the attribute that has the highest information gain. The pseudo code for RDC is shown below.

Input: A relational database D with
a target relation R_t.
Output: A decision tree for predicting

class labels of target tuples.
Procedure: TreeGrowth (T: tree, E: set of
examples, R: relations)
If Stopping_cond ()
then leaf: =CreateNode ();
leaf. label: =Classify ();
return leaf;
else R: =Optimal_refinement ();
T_{Left}: =TreeGrowth (T, R);
T_{Right}: =TreeGrowth (T, R);
Return node (T_{Left},T_{Right}, R);
End

The input to this algorithm consists of the training records E and the relations R. The algorithm works by recursively selecting the best attribute to split the data and expanding the leaf nodes of the tree until the stopping criterions is met. The details of this algorithm are explained below:

The Stopping_cond() function is used to terminate the tree-growing process by testing whether all the records have either the same class label or the same attribute values, and whether the number of records have fallen below some minimum threshold. The Create_Node() function extends the decision tree by creating a new node. The Classify() function determines the class label to be assigned to a leaf node.

The function Optimal_refinement() considers every possible refinement that can be made to the current decision tree and selects the (locally) optimal refinement. Where, TreeGrowth(T, R) denotes the tree resulting from applying the refinement R to the current tree. A relation is active if it appears in the current tree, or it is the target relation. The following algorithm is used to find the optimal refinement to the tree, as shown below.

Input: A relational database D with a target
relation R_t, and the attributes.
Output: The optimal attribute $A_{max}$with the
highest information gain.
Procedure: set R_t to active;
for each active relation P,
A_{max}: = $Find\_Gain_{max}$ ();
for each unactive relation R
for each key/foreign_key k of R
if R can be joined to some
active relation P with $P_{.k}$
then propagate IDs from P to R;
A_{max}: = $Find\_Gain_{max}$ ();
for each foreign_key k'≠k of R
propagate IDs from R to
relation P that is pointed to by R.k
Amax: = Find_Gainmax ();
return A_{max};
End

Where the $Find\_Gain_{max}$() function is to find the best split attribute, and the propagate ID method is explained earlier. The procedure described above is to find the optimal refinement for the tree: for every active relation P, find the best refinement by computing the information gain of every possible attributes; for every unactive relation R that can be joined with some active relation P, propagate IDs from P to R, and find the best split attribute A_{max}. Repeat this procedure until the tree is totally constructed.

2. 4 Information gain

ID3[3] uses information gain as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node N represent or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found. The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} pi \log 2(pi);$$

where pi is the probability that an arbitrary tuple in D belongs to class Ci and is estimated by |Ci,D|/|Dj|. A log function to the base 2 is used, because the information is encoded in bits. Info(D) is just the average amount of information needed to

identify the class label of a tuple in D. Note that, at this point, the information we have is based solely on the proportions of tuples of each class. Info(D) is also known as the entropy of D.

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, fa1, a2, : : : , avg, as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, D1, D2, : : : , Dj, whereDj contains those tuples in D that have outcome aj of A. These partitions would correspond to the branches grown from node N. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class). How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by

$$\mathrm{Info}_A(D) = \sum_{j=1}^{v} |Dj/D| \times \mathrm{Info}(Dj)$$

The term $|Dj/D|$ acts as the weight of the jth partition. $\mathrm{Info}_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions. Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the newrequirement (i.e., obtained after partitioning on A). That is,

$$\mathrm{Gain}(A) = \mathrm{Info}(D) - \mathrm{Info}_A(D)$$

In other words, Gain(A) tells us how much would be gained by branching on A. It is the expected reduction in the information requirement caused by knowing the value of A. The attribute A with the highest information gain, (Gain(A)), is chosen as the splitting attribute at node N. This is equivalent to saying that we want to partition on the attribute A that would do the "best classification," so that the amount of information still required to finish classifying the tuples is minimal (i.e., minimum $\mathrm{Info}_A(D)$).

III. CONCLUSIONS

This paper commences with the comparisons between the 2 pioneer algorithms(MRDTL and MRDTL-2) for multi relational decision tree classification. After that, we present RDC, a new approach for multi-relational decision tree classification. It integrates the multi-relational decision tree and tuple ID propagation. So it can achieve much higher efficiency. These features make it appropriate for multi-relational classification in real world databases. There are several possible extensions to RDC. Currently, RDC uses a simple approach to dealing with missing attribute values in the data. Incorporation of more sophisticated methods into RDC will increase the accuracy of it. Incorporation of sophisticated pruning methods or complexity regularization techniques into RDC will minimize overfitting and improve generalization.

REFERENCES

[1]. Atramentov, A., Leiva, H., and Honavar, V. 2003. A Multirelational Decision Tree Learning Algorithm- Implementation and Experiments, ILP LNCS, Vol.2835, pp. 38-56.

[2]. Leiva, HA. 2002. A multi-relational decision tree learning algorithm, ISU-CS-TR, Lowa State University, pp.02-12.

[3]. Han, J., Kamber, M. 2007. Data Mining: Concepts and Techniques", 2nd Edition, Morgan Kaufmann.

[4]. Xu GM, Yang BR, Qin YQ, " New multi relational naïve Bayesian classifier", Systems Engineering and Electronics, vol. 30, No.4, pp 655-655, 2008.

[5]. Hector Ariel Leiva, "A multi-relational decision tree learning algorithm", M.S. thesis, Department of computer Science, Iowa State University, 2002.

[6]. Knobbe. J, H. Siebes, and Van der Wallen, "Multi-relational Data Mining" , *Proceedings of Benelearn*, 1999.

[7]. Yin X, Han J, and Yu PS, "CrossMine: Efficient

[8]. Classification across Multiple Database Relations". In Proceedings of 20th Int. Conf. on Data Engineering (ICDE'04), 2004.

[9]. Hongyan Liu, Xiaoxin Yin, and Jiawei Han, "An Efficient Multi-relational Naïve Bayesian Classifier Based on Semantic Relationship Graph" , *MRDM-2005*, Chicago, 2005.