Decision Tree in Relational Databases

Alireza Mohammadi
Sahar Nafisi
Data mining Fall 2017
(Course Master: Dr. Vahidipour)

Source Paper: Bentayeb, Fadila, and Jérôme Darmont. "Decision tree modeling with relational views." Foundations of Intelligent Systems (2002): 423-431.

Abstract

- Data mining is a useful decision support technique for discovering production rules in warehouses or corporate data.

- Data mining research has made much effort to apply various mining algorithms efficiently on large databases.
  - Long processing time as a serious problem in such algorithms
  - Integrating data mining methods with the framework of traditional databases
  - Take advantage of the efficiency provided by SQL engines

Abstract (cont.)

- In this presentation, we propose an integrating approach for decision trees within a classical database system.
  - try to discover knowledge from relational databases, in the form of production rules, via a procedure embedding SQL queries
  - Define decision tree by successive, related relational views
- Classical Induction Decision Tree (ID3) algorithm selected to build the decision tree.
- The output of procedure compared with the output of an existing and validated data mining software, SIPINA.

Introduction

- Data mining tool vendors tend to integrate more and more database features in their products
- Long processing time required by data mining algorithms, remains a critical issue.
  - Current systems consume minutes or even hours to answer simple mining queries on very large databases
- Database vendors recently began to integrate data mining methods in the heart of their systems.
- Integrating data mining algorithms within the traditional database systems is one of the key challenges for research in both the databases and data mining fields.

Introduction (cont.)

- There are an impressive amount of work related to association rules, their generalization, and their scalability
- Less work has been done in the context of other classical data analysis techniques from the machine learning field, e.g., clustering or classification.
  - most research focused on scaling data mining techniques to work with large data sets

Introduction (cont.)

- Database vendors developed extensions to SQL and Application Programming Interfaces (APIs).
  - These tools allow client applications to explore and manipulate existing mining models and their applications through an interface similar to that used for exploring tables, views and …
- In the following, we propose to integrate classical data analysis techniques (decision tree-based methods) within relational database systems.

Introduction (cont.)

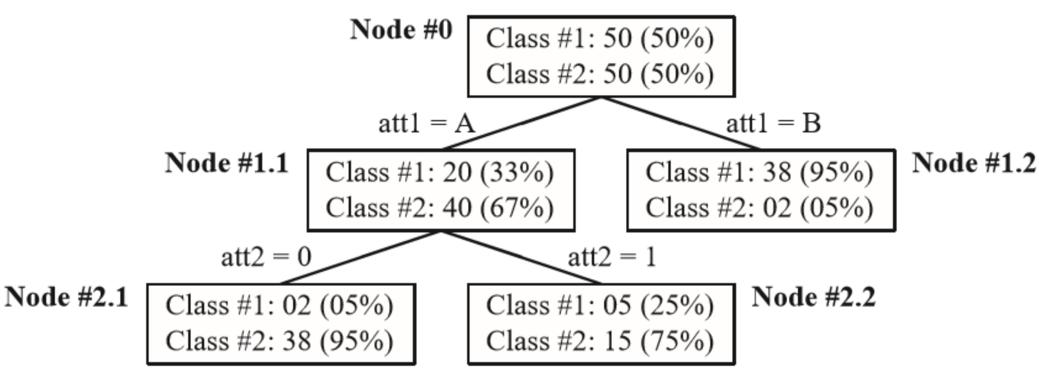
- To achieve this goal, we only use existing structures, namely, relational views.
  - we designed a SQL stored procedure that uses SQL queries to generate the decision tree
- Main differences between our approach and the existing ones:
  - existing methods extend SQL to support mining operators when our approach only uses existing SQL statements and structures
  - existing methods use APIs when our approach does not

Principle of approach

- Induction graphs produce "if-then"-like rules
- They take as input a set of objects (tuples) described by a collection of attributes
- Each object belongs to one of a set of exclusive classes
- A training set of objects whose class (attribute to predict) is known is needed to build the induction graph.

Principle of approach (cont.)

- These methods apply successive criteria on the training population to obtain groups wherein the size of one class is maximized
- This process builds a tree, or more generally a graph
- Figure 1 provides an example of decision tree with its associated rules, where p(Class #i) is the probability of objects to belong to Class #i.



Rule #1: if att1 = A and att2 = 0 then p(Class #2) = 95%
Rule #2: if att1 = A and att2 = 1 then p(Class #2) = 75%
Rule #3: if att1 = B then p(Class #1) = 95%

Fig. 1. Example of decision tree

Principle of approach (cont.)

- The root node of the decision tree is represented by a relational view corresponding to the whole training dataset.

- Each sub-node in the decision tree represents a sub-population of its parent node, we build for each node a relational view that is based on its parent view.

- Then, these views are used to count the population strength of each class in the node with simple GROUP BY queries.

Principle of approach (cont.)

- These counts are used to determine the criteria that helps either partitioning the current node into a set of disjoint sub-partitions based on the values of a specific attribute or concluding that the node is a leaf, i.e., a terminal node.

Principle of approach (cont.)

```
Node #0:   CREATE VIEW v0 AS SELECT att1, att2, class FROM training_set
Node #1.1: CREATE VIEW v11 AS SELECT att2, class FROM v0 WHERE att1='A'
Node #1.2: CREATE VIEW v12 AS SELECT att2, class FROM v0 WHERE att1='B'
Node #2.1: CREATE VIEW v21 AS SELECT class FROM v11 WHERE att2=0
Node #2.2: CREATE VIEW v22 AS SELECT class FROM v11 WHERE att2=1
```

Fig. 2. Relational views associated with sample decision tree

Implementation (Data Structures)

- Stack of nodes
  - A node is contains following fields:
    - num
    - nview
    - rule
    - Entropy
    - population

Implementation (Data Structures)

- List of candidate
  - List of candidates must contain a set of attributes, the information gain associated with these attributes (expressed as a difference in entropy weighted averages), and a list of the nodes that would be generated if the current attribute was selected for splitting the current node.
  - We used a relational table as our principal list, with an embedded table (collection) as the list of nodes.

Implementation (Data Structures)

- List of candidates
  - As a consequence, our table of candidates is composed of the following fields:
    - `Att_name`: considered attribute name
    - `Gain`: information gain
    - `Nodes`: embedded list of associated nodes.

Algorithm

- Input parameters

The input parameters of algorithm are given in this table

Parameter	Name	Default value
Data source table name	table_name	—
Class attribute (attribute to predict)	class	—
Result table name	res_name	BTRES
(Strict) minimum information gain for node building	min_gain	0
Root node view name	root_view	BTROOT
Clean-up views after execution (True/False)	del	TRUE

Table 1. Algorithm input parameters

Algorithm(cont.)

- Pseudo-code

We suppose we can call a procedure named Entropy() that computes both the entropy and the population strength of a node. These data are used when computing the information gain.

```
Create result table
Create root node using the data source table
Compute root node entropy and population strength
Push root node
Update result table with root node
While the stack is not empty do
    Pop current node
    Clean candidate list
    For each attribute but the class attribute do
       Create a new candidate
       For each possible value of current attribute do
          Build new node and associated relational view
          Compute new node entropy and population strength
          Update information gain
          Insert new node into current candidate node list
       End for (each value)
    End for (each attribute)
    Search for maximum information gain in candidate list
    For each candidate do
       If current attribute bears the greater information gain then
          For each node in the list of nodes do
             Push current node
             Update result table with current node
          End for (each node)
       Else
          For each node in the list of nodes do
             Destroy current node
          End for (each node)
       End if
    End for (each candidate)
End while (stack not empty)
```

Fig. 3. Pseudo-code for the **BuildTree** stored procedure

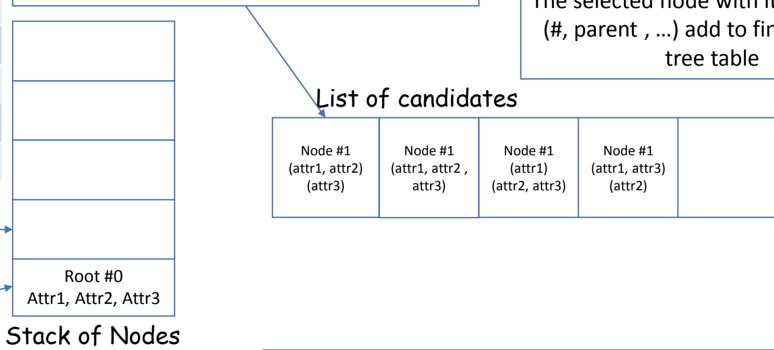
Algorithm Illustration

Attr 1	Attr 2	Attr 3	Class
Low	A	0	#1
High	A	1	#2
Med	B	1	#1
Low	A	1	#2
Med	B	0	#2
High	B	0	#1

2. An element pop from stack and all of possible candidate nodes create from that node and add to candidates list

3. Items in list of candidates sort by a measure (gini, information gain, …) and the best node based on the selected measure add to stack and list be clear.
The selected node with its properties (#, parent , …) add to final decision tree table

Stack pointer



1. At the first, root node add to the stack

4. Step 2, 3 and 4 repeats while there is no node in the stack

Result Output

- The output of our stored procedure, namely a decision tree, is stored into a relational table whose name is specified as an input parameter.
- The table structure reflects the hierarchical structure of the tree.
- Its fields are:
  - node
    - node ID number (primary key)
  - parent, ID number of parent node in the tree (foreign key, references a node ID number); rule, the rule that lead to the creation of this node, e.g., GENDER=FEMALE; and for each value V of attribute E, a field labelled E V, population strength for the considered value of the attribute in this node.

Result Output

LEVEL	NODE	PARENT	RULE	SURVIVOR_NO	P_NO	SURVIVO_YES	P_YES
1	0			1490	68%	711	32%
2	1	0	GENDER=FEMALE	126	27%	344	73%
3	13	1	CLASS=CREW	3	13%	20	87%
3	14	1	CLASS=1ST	4	3%	141	97%
4	21	14	AGE=CHILD	0	0%	1	100%
4	22	14	AGE=ADULT	4	3%	140	97%
3	15	1	CLASS=2ND	13	12%	93	88%
4	19	15	AGE=CHILD	0	0%	13	100%
4	20	15	AGE=ADULT	13	14%	80	86%
3	16	1	CLASS=3RD	106	54%	90	46%
4	17	16	AGE=CHILD	17	55%	14	45%
4	18	16	AGE=ADULT	89	54%	76	46%
2	2	0	GENDER=MALE	1364	79%	367	21%
3	3	2	CLASS=CREW	670	78%	192	22%
3	4	2	CLASS=1ST	118	66%	62	34%
4	11	4	AGE=CHILD	0	0%	5	100%
4	12	4	AGE=ADULT	118	67%	57	33%
3	5	2	CLASS=2ND	154	86%	25	14%
4	9	5	AGE=CHILD	0	0%	11	100%
4	10	5	AGE=ADULT	154	92%	14	8%
3	6	2	CLASS=3RD	422	83%	88	17%
4	7	6	AGE=CHILD	35	73%	13	27%
4	8	6	AGE=ADULT	387	84%	75	16%

Fig. 5. BuildTree result for TITANIC

Conclusion and perspectives

- We presented a different approach for integrating data mining operators into a database system
- Select ID3 method for its simplicity
- Implemented the ID3 method, as a stored procedure that builds a decision tree by associating each node of the tree with a relational view.