ANNALS of the ORADEA UNIVERSITY.
Fascicle of Management and Technological Engineering

DATA MINING ALGORITHMS BASED ON DECISION TREES

Laviniu Aurelian BĂDULESCU
University of Craiova; Faculty of Control, Computers and Electronics, Software Engineering Department, laviniu_aurelian_badulescu@yahoo.com

Keywords: Databases, Knowledge Management, Data Mining, Decision Tree

Abstract. The objective of classification is to use the training dataset to build a model of the class label such that it can be used to classify new data whose class labels are unknown. Decision tree classifiers are relatively fast compared to other classification methods and is easily interpreted/comprehended by humans. Here are presented the most important decision trees based algorithms and their performances: pruning algorithm based on the MDL principle; PUBLIC, who integrates building and pruning phases; SLIQ; SPRINT and RainForest, a unifying framework for decision tree classifiers.

1. INTRODUCTION

Classification is an important problem in the field of Data Mining and Knowledge Management that has been studied extensively over the years. In classification, we are given a set of example records or the input data, called the *training data set*, with each record consisting of several attributes or features. An attribute can be either a *numerical* attribute or a *categorical* attribute. If values of an attribute belong to an ordered domain, the attribute is called a *numerical* attribute (e.g., income, age). A *categorical* attribute, on the other hand, has values from an unordered domain (e.g., type of car, house, job, department name). One of the categorical attributes is designated as the *classification attribute*; its values are called *class labels*. The class label, indicates the class to which each record belongs. The objective of classification is to analyze the input data and to develop an accurate description or model for each class using the features present in the data. Once such a model is constructed, future records, which are not in the training set, can be classified using the model. The objective of classification is to use the training dataset to build a model of the class label such that it can be used to classify new data whose class labels are unknown [2]. The rest of the paper is organized as follows: In Section 2, we formally introduce the problem of decision tree construction. We present a few decision tree based algorithms and their performances in Sections 3-7 and conclude in Section 8.

2. PRELIMINARIES

A decision-tree classifier builds a model by recursively dividing the training set into partitions so that all or most of the records in a partition have the same class label. Most decision-tree classifiers perform classification in two phases: *tree-growing* (or *building*) and *tree-pruning* (Figure 1). In the *tree-growing* phase the algorithm starts with the whole data set at the root node. The data set is partitioned according to a splitting criterion into subsets. This procedure is repeated recursively for each subset until each subset contains only members belonging to the same class or is sufficiently small. In the *tree-pruning* phase the full grown tree is cut back to prevent over-fitting and to improve the accuracy of the tree [12]. An important approach to pruning is based on the minimum description length (MDL) principle [6]. During the growing phase the splitting criterion is determined by choosing the attribute that will best separate the remaining samples of the nodes partition into individual classes. This attribute becomes the decision attribute at the node. Using this attribute a splitting criterion for partitioning the data is defined, which is either of the form $A< v$ $(v \in dom(A))$ for numeric attributes or $A \in V$ $(V \subseteq dom(A))$ for categorical attributes.

```
BuildTree (data set S)
if all records in S belong to the same class
  return
for each attribute A_i
  evaluate splits on attribute A_i
use best split found to partition S into S_1 and S_2
BuildTree (S_1)
BuildTree (S_2)
```

```
PruneTree (node t)
if t is leaf
  return C(S) +1 /* C(S) is the cost of encoding
                    the classes for the records in set S */
minCost_1:= PruneTree (t_1)          /* t_1 , t_2 are */
minCost_2:= PruneTree (t_2)          /* t's children*/
minCost_t:= min(C(S)+1,C_split(t)+1+minCost_1+ minCost_2)
return minCost_t     /* C_split: cost of encoding a split */
```

Figure 1: Algorithms for decision trees

For selecting the best split point several measures were proposed (e.g. ID3 and C4.5 select the split that minimizes the information entropy of the partitions, while SLIQ and SPRINT use the *gini index*). For a data set *S* containing *n* records the information entropy

$$E(S) = -\sum p_i \log_2 p_i \tag{1}$$

is defined as where p_i is the relative frequency of class *i*. For a split dividing *S* into the subsets S_1 (n_1 records) and S_2 (n_2 records) the entropy is

$$E(S_1,S_2) = \frac{n_1}{n}E(S_1) + \frac{n_2}{n}E(S_2). \tag{2}$$

The gini index for a data set *S* is defined as

$$gini(S) = 1 - \sum p_i^2 \tag{3}$$

and for a split

$$gini_{split}(S) = \frac{n_1}{n}gini(S_1) + \frac{n_2}{n}gini(S_2). \tag{4}$$

Once an attribute is associated with a node, it needs not be considered in the node's children. The most time-consuming part of decision tree construction is obviously the splitting point selection. For each active node the subset of data (a *partition*) fulfilling the conjunction of the splitting conditions of the node and its predecessors has to be constructed and for each remaining attribute the possible splits have to be evaluated. Though selecting the best split point based on the measures described above requires no access to the data itself, but only to statistics about the number of records where a combination of attribute value and class label occurs. This information can be obtained from a simple table consisting of the columns *attrib-name*, *attrib-value*, *class-label* and *count*. This structure is described in [1] as *CC table* and in a similar form as *AVC group* (Attribute-Value-Class) in [4].

In decision theory (for example **risk management**), a decision tree is a graph of decisions and their possible consequences, (including resource costs and risks) used to create a plan to reach a goal. Decision trees are constructed in order to help with making decisions.

3. PRUNING ALGORITHM BASED ON THE MDL PRINCIPLE

The basic idea is to think of the decision tree as encoding the class labels of the records in the training database. The MDL principle assume that the best tree is the tree that encodes the records using the fewest bits[6]. Given a node *t*, we need to encode few information. First, we use one bit to encode the type of each node (leaf or internal node). Second, we need to encode, either for an internal node, the cost of encoding the splitting predicate *P(t)* at node *t*: *Cost(P(t))*, or, for a leaf node, the cost of encoding the records in leaf node *t* with *n* records from the training database: *n*E(t)*, where *E(t)* is the entropy of *t*.

For encoding a tree we use the recursive definition of the minimal cost of a node:

- *t* is a leaf node:

$$cost(t) = n * E(t); \tag{5}$$

- *t* is an internal node with children nodes t_1 and t_2. Choice either make *t* a leaf node, or take the best subtrees, whatever is cheaper,

$$cost(t) = min(n * E(t), 1 + cost(P(t)) + cost(t_1) + cost(t_2)) \tag{6}$$

For prune the tree we first construct decision tree to its maximum size; second, we compute the MDL cost for each node of the tree bottom-up; third, we prune the tree bottom-up: *if cost(t)=n*E(t)*, make *t* a leaf node. Resulting tree is the final tree output by the pruning algorithm.[3]

Performance evaluation comparing pruning algorithms. We present results on seven datasets used previously in the STATLOG project [7]. The datasets are available via anonymous FTP from ftp.strath.ac.uk in directory Stams/statlog. Three criteria are used to compare the pruning algorithms: *the error rate* obtained on the test set, *the size of the pruned tree*, and *the execution time* of the algorithm.

Dataset	Testing Error Rates		Pruned-Tree Size		Execution Times	
	C4.5	MDL	C4.5	MDL	C4.5	MDL
Australian	15.5	15.3	38.2	23.6	0.1	0.1
Diabetes	24.7	24.1	65.5	34.8	0.1	0.2
DNA	8.5	8.1	65.0	51.0	4.0	4.0
Letter	15.7	15.8	1266.3	1174.8	24.9	24.6
Satimage	14.8	14.6	244.8	167.0	19.9	19.9
Segment	5.3	5.5	71.0	56.2	3.8	3.7
Vehicle	29.2	29.3	101.2	72.1	0.9	0.9

Table 1. Performance evaluation

Table 1 shows the performance evaluation comparing pruning algorithms. The Testing Error Rates columns show that all the pruning algorithms lead to error rates which are not too different from each other. C4.5 and MDL pruning perform robustly for all the datasets. Next, in Pruned-Tree Size columns, we compare the sizes (in terms of number of nodes) of the pruned trees produced by each of the pruning algorithms. The results show that the MDL pruning algorithm achieves trees that are significantly smaller than the trees generated by the C4.5 pruning algorithm. The results from Execution Times columns shows that C4.5 and MDL have closely values for the execution times for all the datasets.

4. PUBLIC INTEGRATES BUILDING AND PRUNING PHASES

MDL bottom-up pruning requires construction of a complete tree before the bottom-up pruning can start. PUBLIC is a decision tree classifier that during the growing phase, first determines if a node will be pruned during the following pruning phase, and subsequently stops expanding such nodes. The idea of PUBLIC is to prune the tree during - not after - the tree construction phase. This is possible by calculate a lower bound on *cost*(*t*) and compare it with *n*E*(*t*).

Theorem 1. (*Lower Bound*) *Consider a classification problem with k predictor attributes and J classes. Let T_t be a subtree with s internal nodes, rooted at node t, let n_i be the number of records with class label i. Then*

$$cost(T_t) \geq 2 * s + 1 + s * \log_2 k + \sum_{i=s+2}^{J} n_i \, . \tag{7}$$

Lower bound on *cost*(T_t) is thus the minimum of:

(i) (*t* becomes a leaf node) $\quad n * E + 1 \quad (8)$

and

(ii) (subtree at t remains) $$2*s+1+s*\log_2 k+\sum_{i=s+2}^{J} n_i\ . \qquad (9)$$

In PUBLIC, the modified pruning algorithm shown in Figure 2 is invoked from the build procedure periodically on the root of the partially built tree. Note that once the building phase ends, there are no leaf nodes belonging to the ”yet to be expanded” category. As a result, applying the pruning algorithm in Figure 2 at the end of the building phase has the same effect as applying the original pruning algorithm and results in the same pruned tree as would have resulted due to the original pruning algorithm.[9]

```
ComputeCost&PrunePublic(Node $t$)   /* $S$ is the set of data records for $t$ */
if $t$ is a ”yet to be expanded” leaf return lower bound on subtree cost at $t$
if $t$ is a ”pruned” or ”not expandable” leaf return ($C(S)$+1)
$minCost_1$ := ComputeCost&PrunePublic($t_1$);      /* $t_1$ and $t_2$ are $t$’s children */
$minCost_2$ := ComputeCost&PrunePublic($t_2$);
$minCost_t$ := min($C(S)$ + 1, $C_{split}(t)$ + 1 + $minCost_1$ + $minCost_2$);
if $minCost_t$ = $C(S)$ + 1
  Prune child nodes $t_1$ and $t_2$ from tree
  Delete nodes $t_1$ and $t_2$ and all their descendants from $Q$    /* Q is a queue */
  Mark node $t$ as pruned
return $minCost_t$
```

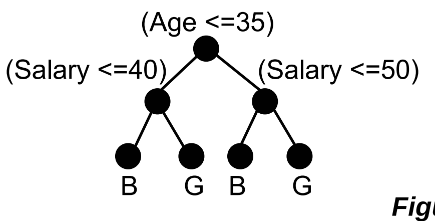
Figure 2. PUBLIC’s Pruning Algorithm

5. SLIQ OR SUPERVISED LEARNING IN QUEST

SLIQ uses a pre-sorting technique in the tree-growth phase to reduce the cost of evaluating numeric attributes. This sorting procedure is integrated with a breadth-first tree growing strategy to enable SLIQ to classify disk-resident datasets. In addition, SLIQ uses

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

TRAINING SET



Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Age List

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

Salary List

	Class	Leaf
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

Class List

AFTER PRE-SORTING

Figure 3. Example of Pre-Sorting

a fast subsetting algorithm for determining splits for categorical attributes. SLIQ also uses a new tree-pruning algorithm based on the MDL principle.[10] This algorithm is inexpensive and results in compact and accurate trees. The combination of these techniques enables SLIQ to scale for large data sets and classify data sets with a large number of classes, attributes, and examples.

Pre-Sorting and Breadth-First Growth. The first technique used in SLIQ is to implement a scheme that eliminates the need to sort the data at each node of the decision tree. Instead, the training data are sorted just once for each numeric attribute at the beginning of the tree growth phase.[5] To achieve this pre-sorting was created a separate list for each attribute of the training data. Additionally, a separate list, called *class list*, is created for the class labels attached to the examples. An entry in an attribute list has two fields, one contains an attribute value, the other an index into the class list. An entry of the class list also has two fields, one contains a class label, the other a reference to a leaf node of the decision tree, The *i*th entry of the class list corresponds to the *i*th example in the training data. Each leaf node of the decision tree represents a partition of the training data, the partition being defined by the conjunction of the predicates on the path from the node to the root. Thus, the class list can at any time identify the partition to which an example belongs. We assume that there is enough memory to keep the class list memory resident. Attribute lists are written to disk if necessary.

Initially, the leaf reference fields of all the entries of the class list are set to point to the root of the decision tree. Then a pass is made over the training data, distributing values of the attributes for each example across all the lists. Each attribute value is also tagged with the corresponding class list index. The attribute lists for the numeric features are then sorted independently. Figure 3 illustrates the state of the data structures before and after pre-sorting.

Tree Pruning. The pruning strategy used in SLIQ is based on the principle of Minimum Description Length (MDL) [10]. There are two components of the pruning algorithm: the encoding scheme that determines the cost of encoding the data and the model, and the algorithm used to compare various subtrees of the initial decision tree *T*.

DATASET	Domain	#Attributes	#Classes	#Examples
Australian	Credit Analysis	14	2	690
Diabetes	Disease diagnosis	8	2	768
DNA	DNA Sequencing	180	3	3186
Letter	Handwriting Recognition	16	26	20000
Satimage	Landusage Images	36	6	6435
Segment	Image Segmentation	19	7	2310
Shuttle	Space Shuttle Radiation	9	7	57000
Vehicle	Vehicle Identification	18	4	846

Table 2. STATLOG Benchmark Datasets and Parameters

Performance Results. Performance evaluation compares SLIQ with the classifiers provided with the IND classifier package [8]. The comparison used datasets from the STATLOG (see Table 2) classification benchmark [7].

In Table 3, the results from Classification Accuracy columns, show that all the three classifiers achieve similar accuracy. However, Pruned-Tree Size columns, shows that there is a significant difference in the sizes of the decision trees produced by the classifiers: C4 produces the largest decision trees for all the datasets. The trees produced by Cart are 2 (Segment) to 16.4 (Australian) times smaller. SLIQ also produces trees that are comparable in size to Cart and 2.1 (Shuttle) to 8.5 (Diabetes) times smaller than C4.

Execution Times columns shows that Cart, which uses cross-validation for pruning, has the largest execution time. The other two algorithms grow a single decision tree, and therefore are nearly an order of magnitude faster in comparison, SLIQ is faster than C4 except for the Australian, Diabetes, and DNA data. The Australian and Diabetes data are very small and,

DATASET	Classification Accuracy			Pruned-Tree Size			Execution Times		
	Cart	C4	SLIQ	Cart	C4	SLIQ	Cart	C4	SLIQ
Australian	85.3	84.4	84.9	5.2	85	10.6	2.1	1.5	7.1
Diabetes	74.6	70.1	75.4	11.5	179.7	21.2	2.5	1.4	1.8
DNA	92.2	92.5	92.1	35.0	171.0	45.0	33.4	9.21	19.3
Letter	84.7	86.8	84.6	1199.5	3241.3	879.0	251.3	53.08	39.0
Satimage	85.3	85.2	86.3	90.0	563.0	133.0	224.7	37.06	16.5
Segment	94.9	95.9	94.6	52.0	102.0	16.2	30.2	9.7	5.2
Shuttle	99.9	99.9	99.9	27	57	27	460	80	33
Vehicle	68.8	71.1	70.3	50.1	249.0	49.4	7.62	2.7	1.8

Table 3. Performance Results

therefore, the full potential of pre-sorting and breadth-first growth cannot be fully exploited by SLIQ. The DNA data consists only of categorical attributes, and hence there are no sorting costs that SLIQ can reduce.

In summary, this set of experiments has shown that Cart achieves good accuracy and small trees. However, the algorithm is nearly an order of magnitude slower than the other algorithms. C4 is also accurate and has fast execution times, but leads to large decision trees. SLIQ, on the other hand, does not suffer from any of these drawbacks. It produces accurate decision trees that are significantly smaller than the trees produced using C4. At the same time, SLIQ executes nearly an order of magnitude faster than Cart.

6. SPRINT. SCALABILITY AND PARALLELIZATION

SPRINT [11] shares with SLIQ the advantage of a one-time sort, but uses different data structures.

Attribute lists. SPRINT initially creates an attribute list for each attribute in the data. Entries in these lists (attribute records) consist of an attribute value, a class label, and the index of the record from which these value were obtained. Initial lists for continuous attributes are sorted by attribute value once when first created. If the entire data does not fit in memory, attribute lists are maintained on disk.

Histograms. For *continuous attributes*, two histograms (C_{above} and C_{below}) are associated with each decision tree node that is under consideration for splitting. C_{below} maintains distribution for attribute records that have already been processed, whereas C_{above} maintains it for those that have not. *Categorical attributes* also have a histogram (*count matrix*) associated with a node, it contains the class distribution for each value of the given attribute.

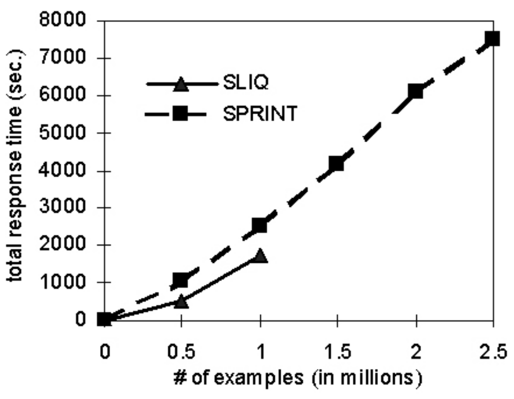


Figure 4. Response times for serial algorithms

SPRINT parallelization focus only on the growth phase due to its data-intensive nature. SPRINT achieves uniform data placement and workload balancing by distributing the attribute lists evenly over *N* processors of a shared-nothing machine. This allows each processor to work on only 1/*N* of the total data. The partitioning is achieved by first distributing the training-set examples equally among all the processors. Each processor then generates its own attribute-list partitions in parallel by projecting out each attribute from training-set examples it was assigned. Lists for categorical attributes are therefore evenly partitioned and require no further processing. However, continuous attribute lists must now be sorted and repartitioned into contiguous sorted sections. The result of this sorting operation is that each processor gets a fairly equal-sized sorted sections of each attribute list.

Serial Performance. (Figure 4) The training sets range in size from 10,000 records to 2.5 million records. For data sizes for which the class list could fit in memory, SPRINT is somewhat slower than SLIQ. In this operating region, we are pitting SPRINT's rewriting of the dataset to SLIQ 's in-memory updates to the class list. What is surprising is that even in this region SPRINT comes quite close to SLIQ. However, as soon as we cross an input size threshold, SLIQ starts thrashing, whereas SPRINT continues to exhibit a nearly linear scaleup.

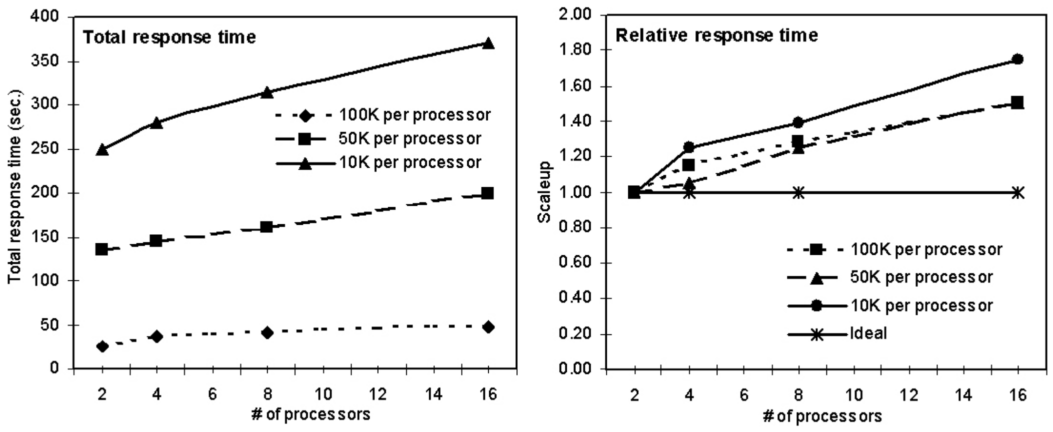


Figure 5. Scaleup of SPRINT

Parallel Performance: Scaleup. (Figure 5) Each processor has a fixed number of training examples and is examined SPRINT's performance as the configuration changed from 2 to 16 processors with 10, 50 and 100 thousand examples on each processor. Since the amount of data per processor does not change for a given experiment, the response times should ideally remain constant as the configuration size is increased. The results show nice scaleup. The drop in scaleup is due to the time needed to build SPRINT's rid hash-tables. While the amount of local data on each processor remains constant, the size of these hash-tables does not. The rid hash-tables grow in direct proportion to the total training-set size. Overall, we can conclude that parallel SPRINT can indeed be used to classify very large datasets.

7. RAINFOREST. A UNIFYING FRAMEWORK FOR DECISION TREE CLASSIFIERS

Is a generic algorithm easy to instantiate with specific algorithms from the literature. The main insight, based on a careful analysis of the algorithms in the literature, is that all algorithms access the data using a common pattern, as described in Figure 6. At the root node *r*, the database is examined and the best splitting criterion *crit*(*r*) is computed. Recursively, at a non-root node *t*, *F*(*t*) (family of a node *t*, the set of tuples of the database that follows the path from the root to *t* when being classified by the tree) is examined and from it *crit*(*t*) is computed. The *AVC-set* of a predictor attribute *a* at node *t* is the projection of *F*(*t*) onto *a* and the class label whereby counts of the individual class labels are aggregated. The *AVC-group* of a node *t* is the set of all *AVC-sets* at node *t*. Note that the size of the *AVC-set* of a predictor attribute *a* at node *t* depends only on the number of distinct attribute values of *a* and the number of class labels in *F*(*t*).

```
BuildTree(Node t, datapartition D, algorithm CL)
Apply CL to D to find crit(t)
let k be the number of children of t
if (k > 0)
  Create k children $c_1$, … , $c_k$ of t
  Use best split to partition D into $D_1$, … , $D_k$
  for (i: =1; i ≤ k; i++)
     BuildTree($c_i$, $D_i$)
```

```
RainForest Refinement:
for each predictor attribute p
   Call CL.find_best_partitioning(AVC-set of p)
k := CL.decide_splitting_criterion();
```

Figure 6: Tree Induction Schema and Refinement [4]

RainForest offers performance improvements of over a factor of five over the SPRINT algorithm, the fastest scalable classification algorithm proposed previously.

8. SUMMARY AND CONCLUSIONS

Decision tree classification is probably the most popular model, because it is simple and easy to understand. Decision tree classifiers are relatively fast compared to other classification methods. A number of the most important decision tree based algorithms for constructing decision trees are presented here, including: pruning algorithm based on the MDL principle; PUBLIC, who integrates building and pruning phases; SLIQ, who uses a pre-sorting technique in the tree-growth phase to reduce the cost of evaluating numeric attributes; SPRINT, remarkably by scalability and parallelization and RainForest, a unifying framework for decision tree classifiers. Classification is an important problem in the field of data mining that has been studied extensively over the years. Data mining involves the development of tools that can extract patterns from large databases and decision tree based algorithms are a powerful such a tools.

BIBLIOGRAPHY

1. Chaudhuri, S., Fayyad, U., Bernhardt, J., *Scalable Classification over SQL Databases*, In Proc. ICDE-99, Sydney, Australia, IEEE Computer Society, 1999, pp. 470–479.
2. Du, W., Zhan, Z., *Building Decision Tree Classifier on Private Data*, Australian Computer Society, Inc., IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining, Maebashi City, Japan, Conferences in Research and Practice in Information Technology, Vol. 14, 2002.
3. Gehrke, J., Loh, W.-Y., *Advances in Decision Tree Construction*, In Proc.KDD-2001.
4. Gehrke, J., Ramakrishnan, R., Ganti, V., *RainForest - A Framework for Fast Decision Tree Construction of Large Datasets*, In Proc. of the 24th VLDB Conference, New York, USA, Morgan Kaufmann, 1998, pp. 416–427.
5. Mehta, M., Agrawal, R., Rissanen, J., *SLIQ: A Fast Scalable Classifier for Data Mining*, In Proc. EDBT'96, Avignon, France, volume 1057 of LNCS, Springer, 1996, pp. 18–32.
6. Mehta, M., Rissanen, J., Agrawal, R., *MDL-based Decision Tree Pruning*, In Proc.KDD-95, Montreal, Canada, 1995.
7. Michie, D., Spiegelhalter, D., J., Taylor, C., C., *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, 1994.
8. NASA Ames Res Ctr. Intro. to IND Version 2.1, GA 23-2475-02 edition, 1992.
9. Rastogi, R., Shim, K., *PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning*, In Proc. of the 24th International Conference on Very Large Data Bases, New York, USA, 1998, pp. 404-415.
10. Rissanen, J., *Stochastic Complexity in Statistical Inquiry*, World Scientific Publ. Co., 1989.
11. Shafer, J., C., Agrawal, R., Mehta, M., *SPRINT: A Scalable Parallel Classifier for Data Mining*, In Proc. VLDB'96, Mumbai (Bombay), India, Morgan Kaufmann, 1996, pp. 544–555.
12. Uwe, K., Dunemann, S., O., *SQL Database Primitives for Decision Tree Classifiers*, CIKM'01 Atlanta, ACM , GA USA 2001.