RESEARCH ARTICLE OPEN ACCESS

Performance Analysis of Decision Tree Classifiers

Md.ghouse Mohiuddin[1], P.Premchand[2]

[1]Department of Computer Science, Palamuru University, Mahboobnagar

[2]Department of Computer science & Eng., Osmania University, Hyderabad

ABSTRACT

Decision tree is considered to be one of the most popular data-mining techniques for knowledge discovery. It systematically analyzes the information contained in a large data source to extract valuable rules and relationships and usually it is used for the purpose of Classification/prediction. In Data mining, Classification of objects based on their features into pre-defined categories is a widely studied problem with rigorous applications in fraud detection, artificial intelligence methods and many other fields. In this paper we reviewed various decision tree algorithms with their limitations and also we evaluated their performance with experimental analysis based on sample data.

Keywords- Classification, Decision tree, ID3, CART, SLIQ, SPRINT, SQL.

I. INTRODUCTION

Databases are rich with hidden information that can be used for intelligent decision making. Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Such analysis can help provide us with a better understanding of the data at large. Whereas classification predicts categorical (discrete, unordered) labels, prediction models continuous valued functions [1].

In classification a model or a classifier is constructed to predict categorical labels such as "safe" or "risky" for loan application data, "Yes" or "No" for market data, or "treatment A" or "treatment B" or "treatment C" for medical data.

How does Classification works? Data Classification is two step process. In the first step, a classifier is built describing predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of data base tuples and their association class labels. Each tuple is assumed to belong to a predefined class as determined by another database attribute called the Class label attribute. The individual tuples making up the training set are referred to as a training tuples and are selected from the data base under analysis. In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects. Because the class label of each training tuple is provided, this step is also known as supervised learning. This first step of classification process can also be viewed as the learning of a mapping or function, Y=f(X), that can predict the associated class label "Y" of a given tuple "X".

In the second step Classification, the model is used for classification. First the predictive accuracy of the classifier is estimated. If the accuracy is considered acceptable, the rules can be applied to the classification of new tuples. The accuracy of a classifier on a given test set is the percentage of the test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple. In this paper we have discussed the decision tree classification algorithms including ID3, CART, C4.5, SLIQ, SPRINT and the performance of these classifiers are analyzed by using the data mining tools Sipina research tool and Rapid Miner 6.0. The rest of the paper includes classification problem and description classification algorithm reviewed from literature in section 2. Section 3 contains the SQL approach of SPRINT algorithm. Section 4 contains the experiments and analysis. Section 5 concludes the paper.

II. RELATED WORK

2. Classification Problem:

The classification problem is one of the most common operations in data mining. Given a data set of N records with at least m+1 fields A1, A2,…., Am and C, the problem is to build a model that predicts the value of the field C (class) given the value of M fields. The classification problem [2] is stated as:

Def: Given a database D={t1,t2,…tn} of tuples (items, records) and a set of classes C={c1,c2,…cn} the classification problem is to define a mapping f:D→C where each ti is assigned to one class. A Class Cj contains precisely those tuples mapped to it, that is Cj={ti/f(ti)=Cj, 1≤i≤n, and ti € d}.

The above definition views classification as a mapping from the database to the set of classes. The classes are predefined, non overlapping and partitioned the entire database. The problem usually implemented in two phases:

1. Create a specific model by evaluating the training data. In this step the training data used as input and the developed model as an output.
2. Apply the model developed in step1 by classifying tuples from the target database.

Decision tree algorithm recursively partitions a data set of records using depth-fist greedy approach [3] or breadth-first approach, until all the data items belong to a particular class are identified. A decision tree structure is made of root, internal and leaf nodes. Most decision tree classifiers perform classification in two phases: tree-growing (or building) and tree-pruning. The tree building is done in top-down manner. During this phase the tree is recursively partitioned till all the data items belong to the same class label. In the tree pruning phase the full grown tree is cut back to prevent over fitting and improve the accuracy of the tree [1] in bottom up fashion. It is used to improve the prediction and classification accuracy of the algorithm by minimizing the over-fitting (noise or much data in training data set) [1]. Decision tree algorithm structure is given in two phases as under:

```
BuildTree (data set S)
if all records in S belong to the same class,
return;
for each attribute Ai
evaluate splits on attribute Ai ;
use best split found to partition S into S1 and S2 ;
BuildTree (S1);
BuildTree (S2);
endBuildTree;
```

Fig. 1 Algorithm for decision tree growth phase

```
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 */
minCost1:= PruneTree (t1);
minCost2:= PruneTree (t2); /* t1, t2 are t's
children*/
minCostt:= min{ C(S)+1, Csplit(t)+1+minCost1+
minCost2 };
return minCostt; /* Csplit: cost of encoding
a split */
EndPruneTree;
```

Fig.2 Algorithm for decision tree prune phase

Decision tree algorithms are implementable in both serial and parallel form. Parallel implementation of decision tree algorithms is desirable in-order to ensure fast generation of results especially with the classification/prediction of large data sets; it is also possible to exploit the underlying computer architecture [17]. However when small medium data sets are involved, the serial implementation of decision tree algorithms is easy to implement and desirable. In the following sections we will briefly discuss the popular decision tree approaches.

2.1 ID3

ID3(Iterative Dichotomized) algorithm[4], is based on information theory and attempts to minimize the expected number of comparisons. The base of ID3 is Concept Learning System (CLS) algorithm. CLS algorithm is the basic algorithm for decision tree learning. The tree growth phase of CLS is the matter of choosing attribute to test at each node is by the trainer. ID3 improves CLS by adding a heuristic for attribute selection. ID3 is based on Hunt's algorithm [3] and is implemented in serially [5]. This algorithm recursively partitions the training dataset till the record sets belong to the class label using depth first greedy technique. In growth phase of the tree construction, this algorithm uses information gain, an entropy based measure, to select the best splitting attribute, and the attribute with the highest information gain is selected as the splitting attribute.

The concept used to quantify information is called entropy. Entropy is used to measure the amount of uncertainty in a set of data. Certainly when all the data in given set belongs to a single class, there is no uncertainty and the entropy is zero. The objective of the decision tree classification is to iteratively partition the given data set into subsets where all elements in each final subset belong to the same class.

For a given set of probabilities p1,p2,…,ps where $\sum_{i=1}^{s} pi = 1$, entropy is defined as

$$\mathbf{H(p1,p2,\ldots,ps) = \sum_{i=1}^{s}(pi\log(1/pi))} \qquad \mathbf{(1)}$$

Given a database state, D, H(D) finds the amount of order in that state. When that state is split into s new states S={D1,D2,….,Ds}. The ID3 algorithm calculates the gain of a particular split by the following formula:

$$\mathbf{Gain(D,S) = H(D) - \sum_{i=1}^{s} P(Di)\,H(Di)} \qquad \mathbf{(2)}$$

ID3 doesn't give accurate result when there is too much noise or details in the training data set, thus an intensive pre-processing of data is carried out before building a decision tree model with ID3[5]. One of the main drawbacks of ID3 is that the measure Gain used tends to favor attributes with a large number of distinct values [1]. It only accepts categorical attributes in building a tree model. This decision tree algorithm generates variable branches per node.

2.2 C4.5

C4.5 algorithm [6], is an improved version of ID3; this algorithm uses Gain Ratio as splitting criteria, instead of taking gain in ID3 algorithm for splitting criteria [3] in tree growth phase. Hence C4.5 is an evolution of ID3 [4]. This algorithm handles both continuous and discrete attributes- In order to handle continuous attributes, C4.5 creates a threshold and then splits the list into those whose attribute value is above the threshold and those that are less than or equal to it [6]. Like ID3 the data is sorted at every node of the tree in order to determine the best splitting attribute. The splitting process stops when the number of instances to be split is less than the threshold. The main advantages of C4.5 is when building a decision tree, C4.5 can deal with datasets that have patterns with unknown attribute values. C4.5 can also deal with the case of attributes with continuous domains by discretization.

This algorithm handles training data with attribute values by allowing attribute values to be marked as missing. Missing attribute values are simply not used in gain and entropy calculations. It has an enhanced method of tree pruning that reduces misclassification errors due to noise or too much detail in the training data set. The Gain ratio used in this approach is defined as:

$$\mathbf{GainRatio(D,S) = Gain(D,S)/H(|D1|/|D|,\ldots,|Ds|/|D|)} \quad \mathbf{(3)}$$

For splitting purposes, C4.5 uses the largest gain ratio that ensures a larger than average information gain.

2.3 CART

CART (Classification and Regression Trees) [1], is a technique that generates a binary decision tree. CART is unique from other Hunt's based algorithm as it is also use for regression analysis with the help of regression trees. The regression analysis feature is used in forecasting a dependent variable (result) given a set of predictor variables over a given period of time. The CART decision tree is a binary recursive partitioning procedure capable of processing continuous and nominal attributes both as targets and predictors [1]. In CART trees are grown, uses gini index for splitting procedure, to a maximum size without the use of a stopping rule and then pruned back (essentially split by split) to the root via cost-complexity pruning. The CART mechanism is intended to produce not one, but a sequence of nested pruned trees, all of which are candidate optimal trees. The CART mechanism includes automatic (optional) class balancing, automatic missing value handling, and allows for cost-sensitive learning, dynamic feature construction, and probability tree estimation [6]. In this algorithm the splitting is performed around what is determined to be the best splitting point, where best is defined by:

$$\mathbf{\Phi(s/t) = 2P_L P_R \sum_{j=1}^{m} |P(C_j / t_l) - P(C_j / t_R)|} \quad \mathbf{(4)}$$

This formula is evaluated at the current node , t, and for each possible splitting attribute and criterion, s. Here L and R are used to indicate the left and right sub trees of the current node in the tree. P_L and P_R are the probability that a tuple in the training set will be on the left or right side of the tree.

2.4 SLIQ

SLIQ [7] (Supervised Learning In Quest) was one of the first scalable algorithms for decision tree induction. This can be implemented in serial and parallel pattern. It is not based on Hunt's algorithm for decision tree classification [3]. It partitions a training data set recursively using breadth-first greedy strategy that is integrated with pre-sorting technique during the tree building phase. SLIQ uses a vertical data format, meaning all values of an attribute were stored as a list, which was sorted at the start of the algorithm. This meant that the attributes need not be sorted repeatedly at each node as was the case in existing algorithms like CART and C4.5. With the pre-sorting technique sorting at decision tree nodes is eliminated and replaced with one-time sort, with the use of list data structure for each attribute to determine the best split point. The calculation of gini index for each possible split point can be done efficiently by storing class distributions in histograms, one per class per node. However SLIQ uses a memory resident data structure called *class list* which stores the class labels of each record. This data structure limits the size of the datasets SLIQ can handle [18]. In building a decision tree model SLIQ handles both numeric and categorical attributes. One of the disadvantages of SLIQ is that it uses a class list data structure that is memory resident thereby imposing memory restrictions on the data. It uses Minimum Description length Principle(MDL)[3] in pruning the tree after constructing it MDL is an inexpensive technique in tree pruning that uses the least amount of coding in producing tree that are small in size using bottom –up technique[3, 1]. The SLIQ decision tree algorithm produces accurate decision trees that are significantly smaller than the trees produced using C4.5 and CART. At the same time, SLIQ executes nearly an order of magnitude faster than CART [1].

2.5 SPRINT: Scalable Parallelizable Induction Of Decision Trees:

SPRINT [8] is the updated version of SLIQ and is meant for parallel implementation. SPRINT is an Algorithm for the induction of decision trees from very large training datasets. This Algorithm can handle categorical and continuous-valued attributes. This Algorithm proposes presorting techniques on disk-resident data sets that are too large to fit in memory. It defines the use of new data structures to facilitate the tree construction. SPRINT uses an attribute list data structures that holds the class and RID (Rowid) information. When a node is split the attribute list is partitioned and distributed among the resulting child nodes accordingly. When a list is partitioned, the order of the records in the list is maintained. Hence the partitioned list does not require resorting. SPRINT was designed to easily parallelized, further contributing to its stability.

A decision tree classifier is built in two phases, Growth phase and the Prune phase. In the Growth phase, the tree recursively partitions the data until each partition has all (most) members of the same class or is sufficiently small which can be threshold set by the user. The form of the split used to partition the data depends upon the type of the attribute used in the split. Splits for as continuous attribute 'A' are the form of value(A) < x , where x is the value in the domain of A. Splits for a categorical attribute 'A' are of the form value(A)€X, where X € domain(A).

Next in the Prune step the dependencies on statistical noise or variation that may be particular to the training set only are removed. This way the problem of over fitting data of the training set is taken care of. Several methods have been proposed for this.

The tree growth phase is computationally much more expensive than pruning since the data is scanned multiple times in the part of the computation. On the other hand pruning just works on the fully grown decision tree. In this chapter we concentrate on the Growth phase. The Tree-Growth Algorithm can be expressed in the following manner:

```
Partition (Data S)
    If (all points in S are of the same class or size of the class < threshold) then
        return;
    for each attribute A do
        evaluate splits on attribute A;
    Use best split found to partition S into S1 and S2;
    Partition (S1);
    Partition (S2);
Initial Call: Partition (Training Data)
```

Fig.3 SPRINT Algorithm

III. SQL APPROACH FOR SPRINT ALGORITHM

Scalable data mining in large databases is one of today's challenges to database technologies. Thus, substantial effort is dedicated to a tight coupling of database and data mining systems leading to database primitives supporting data mining tasks. In order to support a wide range of tasks and to be of general usage these primitives should be rather building blocks than implementations of specific algorithms. In this paper, we describe primitives for building and applying sprint decision tree classifiers.

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 [10] as *CC table* and in a similar form as *AVC group* (Attribute-Value-Class) in [9]. It could be created using a SQL query of the following kind [10]:

```
select 'A1' as aname, A1 as avalue,
C as class, count(*)
from S where condition
group by A1, C
union all
select 'A2' as aname, A2 as avalue,
C as class, count(*)
from S where condition
group by A2, C
union all
...
```

The optimizers of most database systems are usually not able to construct a plan consisting of only a single scan typically at least for each grouping a separate scan is required. Thus computing the statistics table in a single scan would be a good candidate for a classification primitive as already observed in [10].

3.1 Unpivoting the Data Table

To avoid multiple unions, a simple solution could be use to change the physical layout of the table . Consider the table DT and suppose each row in DT (each case) were replaced by a set of rows, one for each attribute-value pair. The new schema for the "unpivoted" table is ***UDT (CaseID, AttributeID, AttributeValue, class)***

Now the query to compute the Cls-Count table is:

```
Select AttributeID, AttributeValue, class, count (*)
From UDT
Where condition
Group by class, AttributeID, AttributeValue
```

Fig. 4 SQL approach for data table

However this optimization comes at a huge increase in the cost of data scan. Let us assume the cost of storing an attribute value is constant 'v' bytes. Also assume each attribute A_i has value density $d_i \leq 1.0$ i.e. that on average, the proportion of cases in which A_i has non-null values is d_i. Let case ID require log (N) bits to represent, where N is the number of records. Similarly, for 'm' attributes, log (m) bits are needed to identify an attribute.

Cost of Scanning DT: is N.m.v since every attribute value occupies space in DT.

Cost of Scanning UDT: is $N(\sum di(\log(N) + \log(m) + v))$

The difference in scan costs is $\delta = N(\sum di(\log(N) + \log(m) + v) – mv)$. Assuming same density, then for all i,

$di = d$, hence $\sum di = md$, yields the simplified form of δ / N (cost per case) $= m(d \log(Nm) + v(d – 1))$

It turns out from the analysis given below that this UDT representation incurrence a huge over head.

Analysis for Dense Data: When the data is fully dense (d=1), $\delta /N = m \log(Nm)$ which shows that the DT representation wins easily.

Analysis for Sparse Data: Clearly UDT representation will begin to win when $d \log(Nm) < v(1 – d)$ $(d\leq1)$. This implies $d < v/ \log(Nm) + v$. This will occur only when the data is at very extreme values of densities. Thus the UDT representation may not be suitable even if the data is fairly sparse.

3.2 Unpivot Operator:

The huge cost of data scan can be reduced if the unpivoted view of the data is created on-the-fly. This can be achieved by extending SQL and introducing an operator UNPIVOT, which give the desired unpivoted view of the data without changing the efficient detailed table DT representation of the data. It essentially takes a row from DT and converts it into its UDT representation which is m rows one for each attribute – value pair. The syntax of the operator is:

```
DT.UNPIVOT(AttrValue for AttrID IN ( A1,…,Am)).
Now the set of sufficient statistics can be extracted using the following query:
Select AttrID, AttrValue, class, count(*)
From DT.UNPIVOT(AttrValue for AttrID IN (A1,…,Am))
Group by AttrID, AttrValue, class;
```

Fig. 5 Using Unpivot operator

Essentially the *DT.UNPIVOT ( )* operator generates a view of DT that is equivalent to UDT. This view can be computed efficiently and need not be materialized. Hence the penalty of the data scan will no longer be incurred and the counts table is efficiently generated. So we notice that making a small extension to SQL has helped in doing classifications within the database system.

3.3 Splitting Point

In order to provide a close integration to the database, the construction of user-defined functions that can be called from within SQL statements is advisable. It decreases the number of calls to the database, and makes our code considerably simpler.

```
Select min (giniSplit (attribute, class, num))
From (Select attribute, class, count (*) num
From mine relation
Group by attribute, class)
```

Fig. 6 SQL approach for calculating Gini index

In the above example, the inner SQL statement sorts and groups the training set based on attribute values. The Group By clause is used instead of a simple Order By to count the number of occurrences for each pair attribute/class, since we cannot split cases having the same attribute value. Hence, the splitting criterion is computed faster than before. When the attribute has too many distinct values, a discretization would be advisable. The final result of this query is the best splitting point of a given attribute. This process is repeated for all other attributes, so that the smallest value is chosen.

3.4 Pruning

During the pruning phase, a user-defined function created based on the original tree is used to classify test cases in parallel. This function, which is called from within a SQL statement returns the number of hits and misses during classification.

```
Select class, Classify (...), from mine relation test
```

Fig. 6 SQL approach for Pruning

3.5 Rule Extraction

Once the classification model is constructed, a set of rules can be derived reading the tree, from top to bottom, until reaching each leaf node. The n rules originally defined (where n represents the number of leaf nodes) are generalized, and some of them may be simplified or even eliminated. As a final result we will reach an ordered rule set, such as:

```
If a > 1 and b <= 5 And c > 2 Then
Return 1
Else
If a <= 1 And d = 'A' Then
Return 0
Else...
Else Return 1; {default class}
End If;
```

As in the pruning case, this rule set is transformed into a PL/SQL function, which can be called from within a SQL statement, enabling the classification of new cases in parallel. This function receives the attributes used during the data mining algorithm and returns the predicted classification.

IV. COMPARATIVE ANALYSIS OF CLASSIFICATION TREE ALGORITHMS:

In this section we will discuss the experimental evaluation of classification tree algorithms including ID3, C4.5, SLIQ, SPRINT, and SQL based SPRINT algorithms, by conducting the experiments on various data sets and data mining software.

4.1Performance Evaluation:

The major factors for evaluating the performance of classifiers is classification accuracy- the percentage of test data that are correctly classified. The other important metrics are classification time and size of decision tree. The ideal goal for decision tree classifiers is to produce compact, balanced and accurate trees with the fewest level, in a short classification time.

4.2 Experiments

In order to evaluate our proposed SQL based approaches, we carried out experiments to compare the above said decision tree algorithms based on their Accuracy, Execution time and Tree Size. This experiment has carried out on different data sets including financial (credit), transportation (vehicle), science, handwriting recognition (letter), medical (diabetes), and mushroom data sets taken from the UCI Machine Learning Repository and we have chosen Sipina & Tanagra Research data mining tools to carry out our experiments.

Table 1 Details of the data sets

Sl.No.	Datasets	#Attributes	#Classes	#Instances
1	Diabetes	09	2	768
2	Vehicle	19	48	846
3	Credit-g	21	2	1000
4	Segment	20	7	2310
5	Mushroom	23	2	8124
6	Letter	17	26	20000

Based on the observations on our experimental results the comparison is as follows:

4.2.1. Accuracy:

Table 2 Representing Accuracy percentages of the different algorithms

Sl. No.	Data Set	ID3	C4.5	SLIQ	SPRINT
1	Diabetes	0.2760	0.2604	0.246	0.249
2	Vehicle	0.3191	0.2979	0.13	0.11
3	Credit-g	0.3100	0.3180	0.09	0.05
4	Segment	0.0117	0.0589	0.03	0.06
5	Mushroom	0.0000	0.0000	6.44	5.13
6	Letter	0.1780	0.1786	0.175	0.15

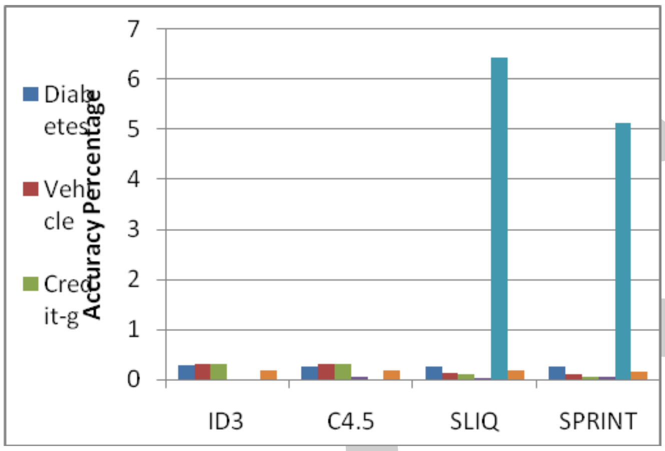

Fig. 7 Showing the Accuracy Percentage of different Classifiers

Accuracy is the reliability of the decision tree and one of the most important parameters which is used for comparing different approaches. This parameter is relevant to the percentage of test samples that are correctly classified. Generally we try to evaluate the classification error rate on the test data set, i.e. the part f the data set which has not been used during the learning process.

The above table 2 shows the cost or error rate percentage of different classifiers. From the results it can be seen that, the lesser percentage of error rate for a classifier is the highest percentage of accuracy of that classifiers. The resulted values specify that the error rate for SLIQ and SPRINT is comparatively lower than the classifiers ID3 & C4.5. The accuracy produced by ID3 & C4.5 is nearly similar with each other, and the accuracy of the scalable classifiers SLIQ & SPRINT are very much similar. In overall comparison we can conclude that SPRINT is efficient and accurate classifiers than other classifiers.

4.2.2. Execution Time

Table 3 Representing the Execution Time of Classifiers

Sl. No.	Data Set	ID3	C4.5	SLIQ	SPRINT
1	Diabetes	234	187	180	265
2	Vehicle	312	280	180	268
3	Credit-g	234	702	766	790
4	Segment	656	390	520	620
5	Mushroom	312	437	400	470
6	Letter	8471	11887	7664	8100

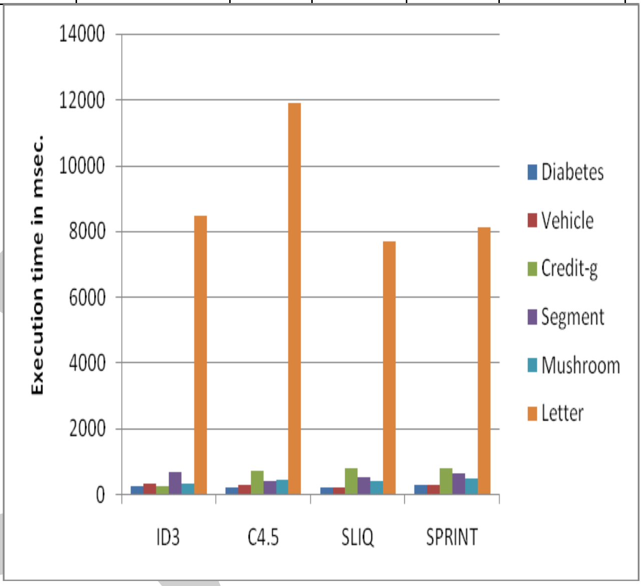

Fig. 8 Comparison of Execution Time of Classifiers

The criterion for comparing the pruning algorithms is the execution time of the algorithms. This parameter is the time which is taken for learning and constructing decision trees. Different approaches try to shorten the time. Table 3 shows the execution times of different Classification algorithms in milliseconds. In our observation the number of instances increases the time taken to construct the decision tree also increases. It can be observed that for the smallest dataset diabetes (768 instances) the execution time of classifier SLIQ is 180 msec., and the highest execution time 11887 msec. is for the largest data set Letter (20000 instances). In overall observation SLIQ is faster than SPRINT and SPRINT is faster than ID3 and C4.5.

4.2.3. Tree Size

Table 4 Representing the Tree Size of Classifiers

Sl. No.	Data Set	ID3	C4.5	SLIQ	SPRINT
1	Diabetes	29	11	21	23
2	Vehicle	37	65	49	47
3	Credit-g	18	173	150	96
4	Segment	75	55	28	37
5	Mushroom	34	30	15	24
6	Letter	1233	2215	879	891

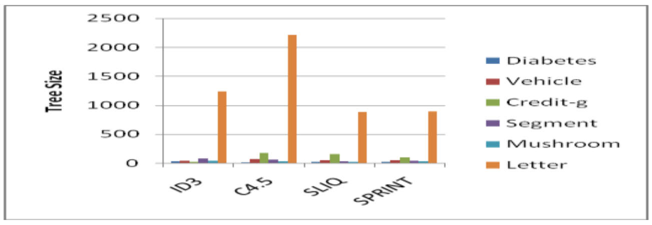

Fig. 9 Comparison of Tree Size of Different Classifiers

Table 4 shows the tree sizes of different decision tree classifiers. The results show that there is a significant difference in the sizes of the decision trees generated by different classifiers. One common fact to be noted is that as the number of instances grows the size of the tree also increases. There is a direct relationship between the size of the tree and the size of the data. From the overall observation we can say that SLIQ generates smaller size trees than the other classifiers, and the other point to be noted here is that, Sprint and SLIQ generates almost similar trees in size.

In summary these sets of experiments shows that C4.5 is faster classifier than other classifiers, but generates large decision trees. SLIQ on the other hand does not suffer from any drawback with respect to accuracy, execution time, and tree size. It generates smaller decision trees and good accuracy percentage. So it is concluded that SLIQ is the good classifier than SPRINT, C4.5 and ID3 in normal data sets. In the next section we will evaluate the performance of SLIQ and SPRINT in large datasets.

4.2.4. Comparative Analysis of SLIQ & SPRINT in Large Data Sets:

In the previous section we have conducted the experiments on the UCI Machine learning data sets, but its large data set contains only 57000 learning examples. Due to the lack of classification benchmark containing large datasets, we have used the synthetic data sets proposed in [14] for all our experiments. Each record in this synthetic data set consists of (9) attributes.

A research under taken by John Sheffer, Agarwal & Mehta from IBM Almaden Research center proved that, Sprint removes all memory restriction that limits existing decision tree algorithms and produces excellent scaling behavior as SLIQ. But taking this conclusion we have implemented the two algorithms in a data mining tool Rapid Miner Studio and recorded the following results.

For our analysis part we compare the execution times of the classifiers SLIQ and SPRINT on training sets of various sizes. We only compare the SPRINT with SLIQ because; in the previous section we have shown that SLIQ has the outstanding performance in all respects with other popular decision classifiers such as ID3, C4.5, and CART etc. For the disk resident data sets which we will be exploring here, SLIQ is the only viable algorithm.

Here we have used training data sets ranging in size 10,000 records to 2.5 million records as proposed in [14]. This range was selected to examine how well SPRINT performs in operating region where SLIQ can and cannot run. In the following table the execution time in Sec(s) are recorded for SPRINT and SLIQ on various size data sets ranging from 0.5 million to 2.5 million.

Sl.No.	**Data Size (million)**	**SLIQ**	**SPRINT**
1	**0.5**	**750**	**1000**
2	**1**	**1750**	**2400**
3	**1.5**		**4200**
4	**2**		**6250**
5	**2.5**		**7650**

Table 5 Representing the Execution Time of SPRINT & SLIQ

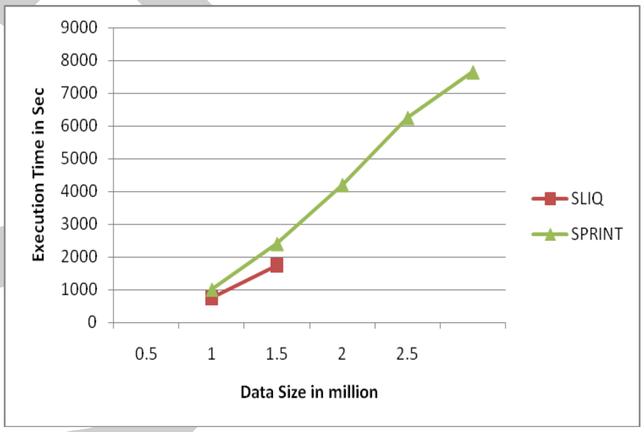

Fig. 10 Comparison of Execution Time of SLIQ & SPRINT

The results recorded in above table are very encouragable. The SLIQ is faster than SPRINT as it consumes less execution time for classification but SLIQ is memory resident. From the above results it can be seen that SLIQ is unable to perform in large data sets, when the data size ranges above 1.5 million, whereas the classifier SPRINT is slower than SLIQ but is more scalable in case of large data sets. From the above results it is observed that SPRINT is performing accurately even when the data set range is exceeding 1.5 million. So SPRINT is the classifier which is scalable in large data sets, and it is the only classifier which is scalable in very large data size where the no other decision tree classifier can compete. This is the fact motivated us to integrate and implement the SPRINT classifier in to the Relational Databases. In the next section we will present the comparative analysis of our proposed classifier SQL based SPRINT and the SPRINT classifier.

4.3 Comparative Analysis of SPRINT & SQL-SPRINT:

There are two important metrics to evaluate the quality of classifiers, classification accuracy and classification time. In the previous section we have compared the results of SLIQ & SPRINT, comparative results shows the improvement of SPRINT over SLIQ. In this section we have conducted the comparative experiments for the performance comparison of our SQL based approach SQL-SPRINT and SPRINT. In this implementation we used a different approach for integrating decision tree method of data mining with DBMS, using only the tools offered by DBMS (Oracle). In this research work, we simply implemented the SPRINT algorithm within Oracle 10g as a PL/SQL stored procedure, by exploiting relational views. We showed that we could process very large databases with this approach without any memory restriction, where the classical in-memory data mining software could not.

4.3.1. Experiments:

In this work we used the classical database methodology of summarization. Like SLIQ and SPRINT we use the same metric (gini index) to choose the best split for each node. We grow our tree in a breadth-first method, and we prune it using the same pruning algorithm. Our classification therefore generate a decision tree identical to the one produced by SLIQ [8] for the same training data set, which facilitates meaningful comparisons of runtime.

For our scaling experiments, we ran our proposed approach on large data sets. The main cost of previous SQL based algorithms was that, they need to access DT – table n-times for each level of the tree growth due to the absence of multiple GROUP BY operator in the previous versions of SQL standard. In this work this deficiency is overcome, and the huge cost of the data scan has been reduced by creating the un-pivoted view of the data on-the-fly. This has been achieved by extending SQL and using the operator UNOIVOT. So the DT-table has been accessed only once regardless of number of attributes.

In this experiment we have used the synthetic data set as proposed in [14]. In this synthetic database each record consists of (9) attributes (salary, commission, age, loan, elevel, car, zipcode, hvalue, hyear). We have conducted our experiments on Client / Server Network System on Windows Platform based Oracle 10g DBMS. We have used training sets with different sizes ranging from 0.5 million to 5 million records. The performance comparison with respect to the execution time of SPRINT and SQL-SPRINT is shown in the following table.

Table 6 Representing Execution Time of SPRINT & SQL-SPRINT

Sl.No.	DATA SIZE	SPRINT	SQL-SPRINT
1	0.5	1000	500
2	1	2300	800
3	1.5	4200	1200
4	2	6250	1500
5	2.5	7600	1800
6	3		2200
7	3.5		2500
8	4		2800
9	4.5		3200
10	5		3700

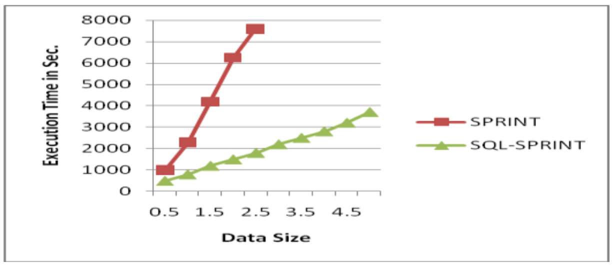

Fig. 11 Representing the Performance of SPRINT & SQL-SPRINT

The experimental results show that the execution time taken by SQL-SPRINT is comparatively less than the time taken by SPRINT for execution. It means that our algorithm SQL-SPRINT is faster than SPRINT classifier. It is also observed that, SQL-SPRINT performs well and continues to scale well in very large databases, whereas the classical SPRINT fails to scale in very large databases. From the above table it can be seen that the classical SPRINT thrashes when the database size increases more than 2.5 million records. Thus we have proved that the performance of our approach SQL-SPRINT is better than classical SPRINT and well scalable in very large databases.

V. CONCLUSION

In this work, we have implemented the classification algorithm SPRINT in DBMS using PL/SQL and SQL statements. In this paper, we have seen a classification Algorithm which can work on a data set of any size. Also we have seen an approach to integrate classification with relational DBMS by extending SQL (UNPIVOT operator). The positive thing of this integration is that it can be used for a variety of splitting indices used within decision tree classifiers. The generic behavior of extension is very important since we need to encapsulate maximum number of mining operations using as few operators as possible.

In order to evaluate our proposed approach SQL SPRINT, we have conducted the experimental evaluation of classification tree algorithms including ID3, C4.5, SLIQ, SPRINT, and SQL- SPRINT algorithms, by conducting the experiments on various data sets and data mining software. From the experiments we have evaluated the results of the classifiers with respect to accuracy, execution time and tree size on various data sets of different sizes. In the comparative study we have compared the performance of SPRINT and SQL-SPRINT on very large databases, and concluded that SQL-SPRINT is scalable on very large databases than SPRINT. It has also proved that SQL-SPRINT is faster than SPRINT.

REFERENCES

[1] Dunham M H (2002), Data Mining : Introductory & advanced Topics, prentice Hall , New Jersey.

[2] Jaiwei Han and Micheline Kamber, Data Mining : Concepts and Techniques, 2/e, Morgan Kaufmann Publishers.

[3] Arun K Pujari , "*Data Mining Techniques*", Universities Press (India) Private Ltd. 2001.

[4] Andrew Colin 1996, Building Decision Trees with ID3 Algorithms, Dr. Dobbs Journal, June 1996.

[5] Becker B. G., *Visualizing Decision Table Classifiers*, Proceedings of Information Visualization, Research Triangle Park, North Carolina, October 1998.
[6] J. Quinlan C4.5 : Programs for Machine Learning, Morgan Kaufmann, 1992.
[7] Metha M., Rissanen J., and Agrawal R., SLIQ: A Fast Scalable Classifier for Data *Mining*, Proc. of the Fifth Int'l Conference on Extending Database Technology, Avignon, France, March 1996.
[8] Shafer J. C., Agrawal R., and Mehta M., A Scalable Parallel Classifier for Data *Mining*, Proc. of the 22nd Int'l Conference on Very Large Databases, Mumbai (Bombay), India, September 1996.
[9] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest – A Framework for Fast Decision Tree Construction of Large Datasets. In A. Gupta, O. Shmueli, and J. Widom, editors, *Proc. VLDB'98, New York, USA*, pages 416–427. Morgan Kaufmann, 1998.
[10] S. Chaudhuri, U. Fayyad, and J. Bernhardt. Scalable Classification over SQL Databases. In *Proc. ICDE-99, Sydney, Australia*, pages 470–479. IEEE Computer Society, 1999. [19]
[11] Sipina research datamining tool available at http://eric-lyon2.fr/~ricco/sipina
[12] Rapid Miner Studio available at http://rapidminer.com/products/rapiminer-studio.
[13] R. Rakotomalala, "Sipina research, http://eric.univ-lyon2.fr/~ricco/sipina.html.
[14] UCI Machine learning Repository (2010) available at http://archive.uci.edu.
[15] *Oracle 9i Data Mining-Concepts*, release 9.2.0.2 . Part No. A95961-02, Copyright © 2002 Oracle Corporation.
[16] Oracle, *Java Stored Procedures Developers Guide*
[17] Gershon N., Eick S. G., and Card S., Information Visualization, ACM Interactions, vol. 5, no. 2, pp. 9-15, March/April 1998.
[18] Chamberlin D., *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann Publishers, Inc, San Mateo, California, 1998