Q5

Understand the working of SLIQ and ARBC classifiers. Give a pseudocode and illustrate the same over a sample dataset of your choice.

1. SLIQ Classifier

Introduction:

Supervised Learning in Quest is a decision tree classifier that can handle both numeric and categorical attributes. 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 a fast sub setting algorithm for determining splits for categorical attributes. SLIQ also uses a new tree-pruning algorithm based on the Minimum Description Length principle. 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.

Pseudo Code:

Key Features,

1. Tree Classifier, handling numeric and categoric attributes
2. Presorting numeric attributes before tree has been built
3. Breadth first growing strategy
4. Goodness test – Gini Index
5. Inexpensive tree pruning algorithm based on Minimum Description Length (MDL)

In Presorting,

1. Eliminate need for sorting data at each node
2. Create sorted list for each numeric attribute
3. Create class list

Split Evaluation,

EvaluateSplits()

For each attribute A do

Traverse attribute list of A

For each value v in attribute list do

Find the corresponding entry in the class list, and hence the corresponding class and the leaf node l

Update the class histogram in leaf l

If A is a numeric attribute then

Compute splitting index for test (A <= v) for l

If A is a categorical attribute then

For each leaf of the tree do

Find subset of A with best split

Update Class List,

UpdateLabels()

For each attribute A used in a split do

Traverse attribute list of A

For each value v in attribute list do

Find the corresponding entry in the class list e

Find the new class c to which v belongs by applying the splitting test at node referenced

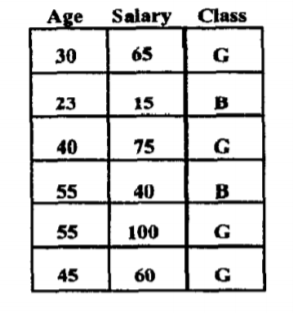
from e

Update the class label for e to c

Update node referenced in e to the child corresponding to the class c

Trace:

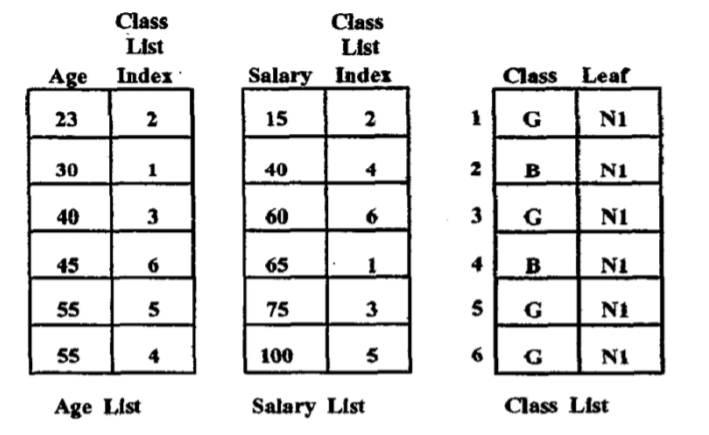
Training Dataset,



Pre-Sorting and Breadth-First Growth,

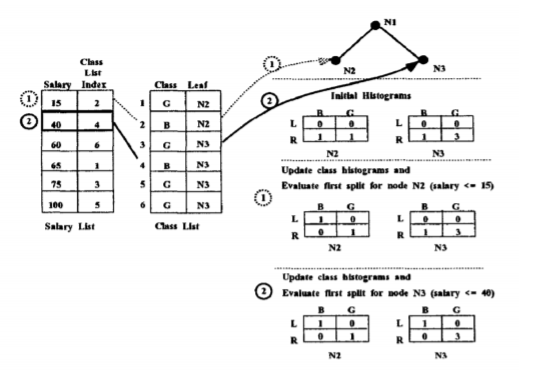
For numeric attributes, sorting time is the dominant factor when finding the best split at a decision tree node. Therefore, the first technique used in SLIQ is to implement a scheme that eliminates the need to sort the data at each node of the decision tree. Instead, the training data are sorted just once for each numeric attribute at the beginning of the tree growth phase. To achieve this pre-sorting, we use the following data structures. We create 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 ith entry of the class list corresponds to the ith 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.

After Presorting,



Processing Node Splits,

Rather than using a depth-first strategy used in the earlier decision-tree classifiers, we grow trees breadth-first. Consequently, splits for all the leaves of the current tree are simultaneously evaluated in one pass over the data. Figure 4 gives a schematic of the evaluation process. To compute the gini splitting-index for an attribute at a node, we need the frequency distribution of class values in the data partition corresponding to the node. The distribution is accumulated in a class histogram attached with each leaf node. For a numeric attribute, the histogram is a list of pairs of the form ¡class, frequency¿. For a categorical attribute, this histogram is a list of triples of the form ¡attribute value, class, frequency¿. Attribute lists are processed one at a time (recall that the attribute lists can be on disk). For each value IJ in the attribute list for the current attribute A, we find the the corresponding entry in the class list, which yields the corresponding class and the leaf node. We now update the histogram attached with this leaf node. If A is a numeric attribute, we compute at the same time the splitting index for the test A 2 v for this leaf. If A is a categorical attribute, we wait till the attribute list has been completely scanned and then find the subset of A with the best split. Thus, in one traversal of an attribute list, the best split using this attribute is known for all the leaf nodes. Similarly, with one traversal of all of the attribute lists, the best overall split for all of the leaf nodes is known. The best split test is saved with each of the leaf nodes. For our example

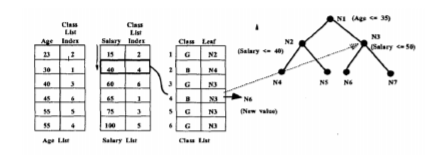


This figure illustrates the evaluation of splits on the salary attribute for the second level of the decision tree. The example assumes that the data has been initially split on the age attribute using the split age 2 35. The class histograms reflect the distribution of the points at each leaf node as a result of the split. The L values represent the distributions for examples that satisfy the test and R values represent examples that do not satisfy the test. We show how the class histograms are updated as each split is evaluated. The first value in the salary list belongs to node N2. So the first split evaluated is (salary ≤ 15) for N2. After this split, the corresponding example (salary 15, class index 2) which satisfies the predicate belongs to the left branch and the rest belong to the right branch. The class histogram of node N2 is updated to reflect this fact. Next, the split (salary 5 40) is evaluated for node N3. After the split, the corresponding example (salary 40, class index 4) belongs to the left branch and the class histogram of node N3 is updated to reflect this fact.

Updating the class list,

The next step is to create child nodes for each of the leaf nodes and update the class list.

Class list updating,



As an illustration, above figure shows the class list being updated after the nodes N2 and N3 have been split on the salary attribute. The salary attribute list is being traversed and the class list entry (entry 4) corresponding to the salary value of 40 is being updated. First, the leaf reference in the entry 4 of class list is used to find the node to which the example used to belong (N3 in this case). Then, the split selected at N3 is applied to find the new child to which the example belongs (N6 in this case). The leaf reference field of entry 4 in the class list is updated to reflect the new value. This is how we create the decision tree.

1. ARBC Classifier

Introduction:

Association rule mining was introduced as a way to find associative patterns from market basket data. The market basket data consist of transactions where a transaction is a set of items purchased by a customer. The motivation for applying this data mining approach on market basket data was to learn about buying patterns and use that information in catalog design, and store layout design. Since then, association rule mining has been studied and applied in many other domains (e.g. credit card fraud, network intrusion detection, genetic data analysis). In every domain, there is a need to analyze data to identify patterns associating different attributes. Association rule mining addresses this need. Many association rule mining algorithms have been proposed in the data mining literature. Apriori and FP-growth are two of them.

Pseudo Code:

Begin

minConfidence

rules = []

freqItemsets = []

support = UpperBoundSupport

while (support LowerBoundSupport AND rules.size < minNumberOfRules) do

L1 = {1 – item itemsets}

For (k=2; Lk1 6= ) do

Ck = generateCandidates (L k 1)

Lk = evaluateCandidates (Ck)

freqItemsets L(k)

end for

maxFreqItemsets = genMaxFreqItemset (freqItemsets)

rules = GenerateAllRules(maxFreqItemsets, minCondfidence)

support = support – delta

freqItemsets = []

end while

R = rules

R = sort(R)

For each rule r ∈ R in sequence do

Temp = ∅

For each instance d ∈ D do

If d satisfies the conditions of r then

Store d.id in temp and mark r if it correctly classifies d

End if

End for

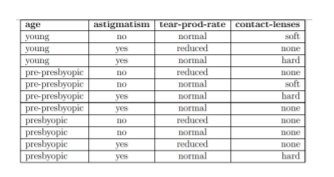
Find the first rule p in C such that Cp, the rules in C up to p, has lowest number of errors and drop all the rules

Add the default class associated with p to the end of C, and return C

End

Trace:

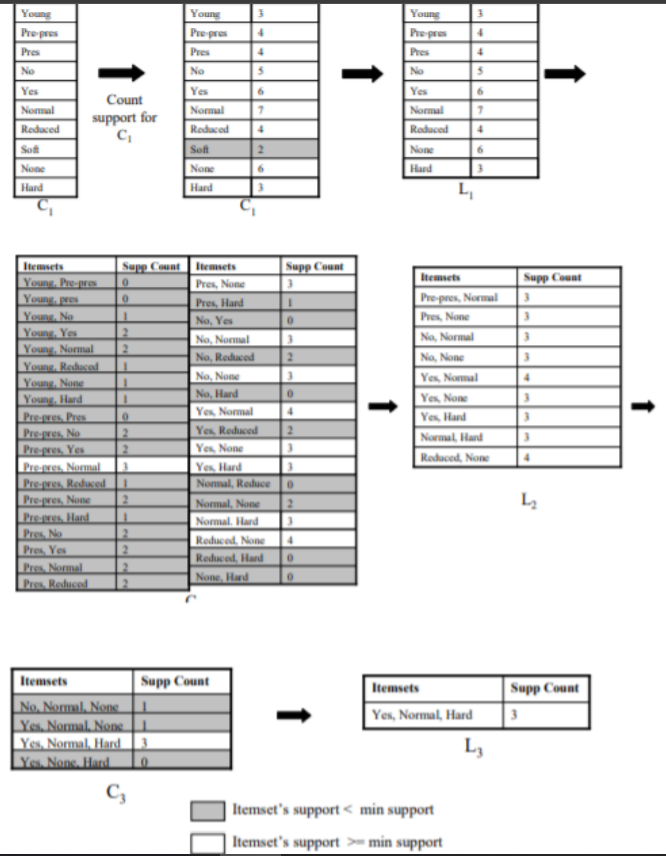
Training Dataset



First, Create Associate Rules,

For this, we can use any associate rule mining algorithm to mine the classification rule. In associative classification, the focus is to produce association rules that have only a particular attribute in the consequent. These association rules produced are called class association rules (CARs). Associative classification differs from general association rule mining by introducing a constraint as to the attribute that must appear on the consequent of the rule. The CBA-RG algorithm is an extension of the Apriori algorithm. The goal of this algorithm is to find all rule items of the form < condset, y > where condset is a set of items, and y ∈ Y, where Y is the set of class labels. The support count of the rule item is the number of instances in the data set D that contain the condset and are labeled with y. Each rule item corresponds to a rule of the form: condset → y.

Applying Apriori with support 3,



Classifying based on rules,

Rule items that have support greater than or equal to minsup are called frequent rule items, while the others are called infrequent rule items. For all rule items that have the same condset, the one with the highest confidence is selected as the representative of those rule items. The confidence of rule items is calculated to determine if the rule item meets minconf. The set of rules that is selected after checking for support and confidence is called the classification association rules (CARs). Given a model and a new instance whose class is unknown, the problem of predicting the instance’s class using the model is an interesting problem. There is more than one way to use the model to predict the instance’s class. In association rule-based classification models, rules in the model are ordered as follows:

* if rule ri has greater confidence than rj, then ri precedes rj, or
* if ri has the same confidence as rj, then the rule with greater support precede the other, or
* if ri has the same support and the same confidence as rj, then the rule with then smaller number of items in the antecedent will precede, or
* if ri has the same support, confidence and antecedent size as rj, then the order between the two rules is random.

Rules of high confidence are thought to be good for classification. Confidence alone may not make a rule very good. For instance, a rule from an instance that appears only once (high confidence but low support) may not be a good rule for classification. Rules with very high confidence and low support are useful in identifying rare events. After Applying apriori, the rules we get are,

Apriori rules with confidence 0.5,

