Classical and Incremental Classification in Data Mining Process

Ahmed Sultan Al-Hegami

Sana'a University, Sana'a, YEMEN

Summary

Knowledge Discovery in Databases (KDD) is an iterative and multi step process that aims at extracting previously unknown and hidden patterns from a huge volume of databases. Data mining is a stage of the entire KDD process that involves applying a particular data mining algorithm to extract an interesting knowledge. One of the important problems that are used by data mining community is so-called classification problem. In this paper we study the classification task and provide a comprehensive study of classification techniques with more emphasis on classical and incremental decision tree based classification. While studying different classification techniques, we provide many important issues that distinguish between each classifier such as splitting criteria and pruning methods. Such criteria lead to the variation of decision tree based classification.

Key words:
Knowledge Discovery in Databases (KDD), Data Mining, Incremental Classifier, Decision Tree, Pruning Technique, Splitting Technique.

1. Introduction

Classification is an important data mining task that analyses a given training set and develops a model for each class according to the features present in the data. The induced model is used to classify unseen data tuples. There are many approaches to develop the classification model including decision trees, neural networks, nearest neighbor methods and rough set-based methods [1,2].

Classification is very important when studying learning strategies, that is, by describing the task of constructing class definition, future data items can be classified by determining if they follow the definition learned [3]. It is particularly useful when a database contains examples that can be used as the basis for decision making process such as assessing credit risks, for medical diagnosis, or for scientific data analysis. Examples of classification task include [4].

- Determining which home telephone lines used for Internet access.
- Assigning customers to predefined customer segments.
- Classifying credit applicants as low, medium, or high risk.
- Assigning keywords to articles as they come in off the news wire.

These applications make use of several products that are available in the marketplace. AC2 from Isoft, is a very well known tool. SPSS is a product based on the tool, called SI-CHAID. Many tools are also used in many data mining packages that combine a variety of approaches, including IBM's Intelligent Miner, Clementine, Thinking Machine's Darwin, and Silicon Graphic's Mineset. Angross has produced a decision tree based analysis system, called KnowledgeSEEKER [5]. This system is a comprehensive program for classification tree analysis. It uses two well-known decision tree tools called CHAID and CART. The wide application and great practical potential of classification has been shown by these applications, which have produced useful results.

Decision tree induction is one of the most common techniques to solve the classification problem [2,6]. It consists of nodes, branches, leaf nodes, and a root. To classify an instance, one starts at the root and finds the branch corresponding to the value of that attribute observed in the instance. This process is repeated at the sub tree rooted at that branch until a leaf node is reached. The resulting classification is the class label on the leaf. The main objective of a decision tree construction algorithm is to create a tree such that the classification accuracy of the tree, when used on unseen data, is maximized. Other criteria such as tree size and tree understandability may also be used.

Many decision tree induction algorithms have been proposed based on different attribute selection and pruning strategies. These methods partition the data recursively until all tuples in every partition have the same class value. The result of this process is a tree that is used for the prediction of future unseen data.

Decision tree induction algorithms operate in two phases, the C*onstruction phase* and *Pruning phase.* The *construction phase* of decision tree usually results in a complex tree that often overfits the data. This reduces the accuracy when applied to unseen data. The *Pruning phase* of decision tree is the process of removing some non-promising branches to improve the accuracy and performance of the decision tree.

Manuscript received December 5, 2007
Manuscript revised December 20, 2007

One of the main drawbacks with the traditional tree induction algorithms is that they do not consider the time in which the data arrived. Researchers have been strongly motivated to propose techniques that update the classification model as new data arrives, rather than running the algorithms from scratch [1,23,24,25], resulting in incremental classifiers. The incremental classifiers that reflect the changing data trends are attractive in order to make the over all KDD process more effective and efficient. In this paper a comprehensive and comparative analysis of traditional and incremental learning algorithms with more emphasis on tree induction approaches and the different splitting and pruning strategies.

2. Classical Tree Induction

Many traditional algorithms for inducing decision trees have been proposed in the literature (e.g., C4.5 [7], CART [9], SPRINT [10], PUBLIC [11], and BOAT [12], based on different attribute selection and pruning strategies. Some of commonly used splitting criteria include *entropy* or *information* gain [6, 7], *gain ratio* [7], *Gini index* [9], *Towing rule* [9], χ^2 and its variant forms [15], S*ummInority* [14]. A detailed survey of different selection techniques can be found in [15,16].

There are two approaches of tree pruning, pre-pruning and post-pruning. In pre-pruning approach, a tree is pruned by stopping its construction by deciding not to further partition the subset of training data at a given node. Consequently, a node becomes a leaf that holds a class value with the most frequent class among the subset of samples. Pre-pruning criteria are based on *statistical significance* [17], *information gain* [18], or *error reduction* [15,20]. Post-pruning removes branches from the completely grown tree, by traversing the constructed tree and uses the estimated error to decide whether some undesired branches should be replaced by a leaf node or not [7,21]. This replacement is the key issue of many pruning criteria that appear in the literature.

Several post-pruning techniques have been proposed based on *cost-complexity* [9,21], *reduced-error* [18,21], *pessimistic-error* [7,18,21], *minimum-error, critical value* [21] and *Minimum Description Length (MDL)* [22]. The objective of such criteria is to find simple and comprehensible tree with acceptable accuracy. A detailed survey of different pruning techniques can be found in [21].

One of the most popular classical decision tree based classifiers is ID3 algorithm. ID3 is an extension to an earlier decision tree based classifier called CLS (Concept Learning System) [13]. CLS uses a look ahead approach when selecting attribute value for a particular node. It explores the space of possible attribute values up to some depth and chooses the best attribute. CLS is computationally expensive because it explores all possible decision trees up to particular depth. Although CLS is not an efficient decision tree classifier, it was the father of ID3 algorithms.

ID3 is a divide-and-conquer approach to decision tree induction, sometimes-called top-down induction of decision tree, was designed by Ross Quinlan [6, 7]. The key success of ID3 lies in its information formula. The goal of this formula is to minimize the expected number of tests to classify an object. A decision tree can be regarded as an information source. For a given object, it generates a message which is a class corresponding to that object. The criterion of selecting an attribute in ID3 is based on the assumption that the complexity of the decision tree is related to the amount of information conveyed by this message [6, 7].

The information formula is applied to training examples in order to select an attribute, which is split best among all other attributes regarding the class value. Once an attribute has the highest information gain it is selected to be the root node of the tree. If the samples have the same class value, then the node becomes a leaf and labeled with that class, otherwise, branches are created from a node represent the data values of that node. Each branch is examined in order to determine if it leads to a leaf node. At this point, a threshold value may be introduced. A threshold is a value that represents the percentage of tuples that have to match the class value. If in a particular branch, the required tuples in the training set has the same class value, a leaf node is created. In case of the threshold value is not maintained, the information formula is again applied to the training set, only those tuples that match the branch value, to determine the next node for the split. This process continues partitioning the training set recursively until either all tuples for a given node belong to the same class or there are no remaining attributes on which the samples may be further partitioned. In the later case, majority voting [1] can be applied. Majority voting involves converting the given node into a leaf that holds the class, which has majority among samples.

Once the decision tree is created, it becomes simple to provide the user with all the rules generated, simply by traversing the tree from the root to the leaf nodes. Each path in the tree represents a rule that classifies the dataset.

When the sets of rules have been obtained from the decision tree based classifier, the rules are evaluated to measure their correctness to avoid the problem of over-fitting [1,2,7]. The overfitting problem results in reduction of the predictive accuracy of the model. The predictive

accuracy of the algorithm can be measured using the training set and verifying the results using a testing set. This method used by Quinlan and called train/test method [6].

Due to some of the limitations of ID3, Quinlan has established an extension to it. He provided a more effective algorithm, C4.5. Generally, ID3 is prone to create very large decision tree, which can be difficult to understand. C4.5 attempts to reduce the size of the decision tree by using a number of methods. Pruning method is one of the techniques used by C4.5 in order to reduce the size of decision tree. Many pruning techniques have been used by C4.5 (i.e. *reduced error pruning* and *pessimistic-error pruning*) that reduce the tree size. Some algorithms look ahead to see if pruning is beneficial and decide whether to prune or not based on some criteria. C4.5 uses an alternative method. It goes ahead; and over-fits the data and then prune. Although this approach is considered to be slower, it is more reliable [7]. Another feature used in C4.5 is that, it combines rules in the pruned tree in order to keep the number of rules minimum.

It has been noticed that, a large training dataset is not the only reason for a large decision tree. An attribute, which has many different values, creates a large number of branches particularly when the attributes are numerical. C4.5 solves this problem by grouping attribute values to keep the number of branches smaller. For example, if a particular attribute has 100 different values, 100 branches will be created for the node uses this attribute, this results in very huge decision tree. In fact, this is one of the disadvantages of ID3. In C4.5, it provides a facility to use ranges as branch values, that is, instead of having 100 branches, there could be only three, a branch where all values are < some value n, = some value n, or a branch where all values are > n.

Another important improvement to C4.5 is the way of splitting the dataset. It has been shown that the information gain criteria are biased in that it prefers the attributes, which have many values. Many alternative approaches have been proposed, such as gain ratio [7], which considers the probability of each attribute value.

3. Incremental Tree Induction

One of the main drawbacks with the classical tree induction algorithms is that they do not consider the time in which the data arrived. Researchers have been strongly motivated to propose techniques that update the classification model as new data arrives, rather than running the algorithms from scratch [1,23,24,25], resulting in incremental classifiers. The incremental classifiers that reflect the changing data trends are attractive in order to make the over all KDD process more effective and efficient.

Incremental algorithms build and refine the model as new data arrive at different points in time, in contrast to the traditional tree induction algorithms where they perform model building in batch manner. Incremental classifiers are widely used techniques that the recognition accuracy of a classifier is heavily incumbent on the availability of an adequate and representative training dataset. Acquiring such data is often tedious, time-consuming, and expensive. In practice, it is not uncommon for such data to be acquired in small batches over a period of time. A typical approach in such cases is combining new data with all previous data, and training a new classifier from scratch. This approach results in loss of all previously discovered knowledge. Furthermore, the combination of old and new datasets is not even always a viable option if previous datasets are lost, discarded, corrupted, inaccessible, or otherwise unavailable. Incremental classifier is the solution to such scenarios, which can be defined as the process of extracting new patterns without losing prior knowledge from an additional dataset that later becomes available.

The problem of dataset over evolving time has motivated development of many incremental classifiers including COBWEB [26], ID4 [27], ID5 [24], ID5R [25] and IDL [28]. The advantages of incremental techniques over traditional techniques are elaborated in [25].

The ID3 algorithm was extended to accommodate incremental learning by several algorithms that were proposed with some degree of ID3-compatibility. An incremental classifier can be characterized as ID3-compatible if it constructs almost similar decision tree produced by ID3 using all the training set. This strategy is maintained by classifiers such as ID4 [27], ID5 [24] and ID5R [25]. These classifiers have a property that they maintain counters at each node to keep track of the examples that have been examined at that node, without retaining these past examples. The counters also help to show how the untested attributes would split the training examples at a particular node.

ID4 [27] was the first ID3-variant to construct the incremental learning. ID4 builds the same tree as the basic ID3 algorithm, when there is an attribute at each decision node that is the best among other attributes. When the relative ordering of the possible test attributes at a node changes due to new incoming examples, all subtrees below that node are discarded and have to be reconstructed. Sometimes, despite training, the relative ordering does not stabilize and therefore results in the decision tree being rebuilt from scratch every time a new

training instance is presented. This thrashing effect was too much of a bottleneck to allow practical applications of ID4, as it effectively discards all previous learning efforts.

ID5 [24] expanded this idea by selecting the most suitable attribute for a node, while a new instance is processed, and restructuring the tree, so that this attribute is pulled-up from the leaves towards that node. This is achieved by suitable tree manipulations that allow the counters to be recalculated without examining the past instances.

In [30] a case was put forward for decision trees, which suppress redundant information. Although the observations made are applicable to incremental learning, no algorithm was given and the authors expressed their reservations about the wider practicality of their ideas. Nevertheless, the paper describes concisely the concepts of tree manipulation and transposition that make ID5 and ID5R powerful.

A blend of the above ideas is also present in the *IDL* algorithm [28]. The notion of topological relevance was introduced as a measure of the importance of an attribute for determining the class of an example. Topological relevance can be calculated in a bottom-up fashion and a decision tree is topologically minimal with respect to the training set, if it satisfies some measure of topological relevance among all attributes and all examples. Incremental induction is not carried out by using a statistical measure, but by trying to obtain a topologically minimal tree. The algorithm achieves impressive results in keeping the tree size considerably lower than ID5R, but can come across severe problems of non-convergence to a final tree form.

From a different point of view, [29] proposed a measure of statistical significance of impurities of nodes to allow CART [1,9] to be used incrementally.

4. Splitting Techniques

Selecting the test attribute at each node of decision tree is one of many reasons that lead to the variation of decision tree algorithms. Various splitting criteria were proposed and used in different decision tree algorithms, including *entropy or information gain* [6], *Gini index* [7], *Towing rule* [9], χ^2 *and its variant forms* [15], *deviance* [19], S*ummInority* [14].

Although one may think that, the choice of evaluation function has an important effect on the accuracy of decision trees, the attribute selection metric, or splitting criterion, has no significant effect on the accuracy of the induced tree [21]. But, most of recent work on splitting criteria by [16] improves purely theoretical attempts to address a problem noticed by [33,34], namely that, the standard information gain formula is biased towards selecting attributes which have many values. In the following subsections we discuss some common attribute selection criteria.

We start our description by specifying the notation to be used in this section. Consider a K-class, N-point dataset at a given node T, which is about to be split into two nodes, T_L and T_R (for left and right) with the proportions of data points, P_l and P_r respectively. The class of each data point is an outcome of discrete random variable, X, which takes values from a set of K class labels, $\{c_1,\ldots,c_k\}$. The probability distribution of X is expressed as $p(X=c_j) = p_j$, where j= l,..,k and $\sum_{j=1}^{k} p_j = 1$. Note that, in each of the following criteria, we provide only the definitions of the measures.

When applied to data splitting, what often evaluated are, the changes in the values of these measures due to the partitioning of the data. Normally, a splitting criterion selects the split that maximizes the amount of gain in a goodness measure or reduction in an impurity measure. The impurity-based measures mean that, after each split; the data of the child nodes are more homogeneous (purer) in terms of class than the data in the parent node.

4.1 Entropy or information gain

The use of information gain as a splitting criterion is popularized by Quinlan [6,7]. Quinlan has used this measure in learning systems called ID3 and C4.5 systems. The entropy of a random variable X is defined as:

$$H(X) = \sum_{j=1}^{k} p_j \log_2 \frac{1}{p_j} = -\sum_{j=1}^{k} p_j \log_2 p_j \quad \text{(define 0 log 0=0)}$$

The value of the entropy attains its minimum, 0, when any p_j=1 (j=1,..,k) (which implies all other p_j's are 0); and the value reaches its maximum, $\log_2 k$, when all p_j's are equal to 1/k. This property is consistent with that desired by an impurity measure: when applied to partitioning data, a split that reduces the entropy of the data also reduces the impurity of the data.

4.2 Gini Index

The measure was introduced by [9], and has been implemented in CART. It has the form:

$$G(X) = \sum_{j \neq 1} p_j p_i = 1 - \sum_{j=1}^{k} p_j^2$$

The Gini Index is another popular splitting criterion that

possesses the desired property of an impurity measure.

4.3 Towing Rule

This measure was also introduced by [9] and has been used and implemented in CART learning algorithm. The towing function can by defined as:

$$\Phi = \frac{P_L P_R}{4}\left[\sum_{j=1}^{k} | P(c_j \setminus T_L) - P(c_j \setminus T_R) |\right]^2$$

Where $P(c_j \setminus T_L)$ and $P(c_j \setminus T_R)$ are proportions of data points in T_L and T_R that belong to class c_j. The towing rule is more appropriate for data, which has a large number of different classes.

4.4 Chi-Squared ($\chi 2$) and its variant forms

This measure is used as splitting criterion in CHAID. It is more error-based than impurity–based. The measure has different variants include that proposed by [15]. The chi-squared criterion is not as widely used in decision tree systems as the previously mentioned measures.

4.5 Deviance

This criterion was proposed by [19] and implemented in S-plus. The deviance function is defined as:

$$D(y_i) = -2\sum_{j}^{k} y_{ij} \log(p_j)$$

Where y_{ij} (i=1,…,n; j=1,…,k) is the i^{th} observation of a K-component random vector Y, whose value takes the form Y= (0,0,…,1_{jth},…,0), denoting that the class of the observation is C_j ; and p_j is as defined earlier in this section. Note that the random vector Y is a different representation of the random variable X described at the beginning of this section; therefore, the deviance is basically the same as the entropy measure. The deviance is the form of a likelihood ratio statistic (and Y follows a multinomial distribution), which is more acceptable to the statistics community. The entropy, instead, is a measure of the average amount of information (in number of bits) needed to convey a message (or, to identify the class of a data point, from the decision tree point of view).

4.6 Summinority

This measure was first used in [14] although the idea does not appear to be new. The idea is that, the most frequent class in a dataset is called the majority class, and all other classes are minority ones. The Summinority measure is simply the sum of the numbers of all minority cases in T_r and T_l. The criterion then selects the split that minimizes this measure. The Summinority is basically an error measure, since as described earlier; decision trees classify a data point based on the majority class at a leaf.

Many imperial studies have been conducted to evaluate the quality of various splitting criteria. These studies have shown that, on average, the Entropy, Gini Index and Towing Rule perform relatively better, while error based criteria, such as Summinority and some χ^2 variants are somewhat less important.

5. Pruning Techniques

Usually, the process of constructing a decision tree leads to generating many branches that may reflect anomalies in the training data due to noise or outliners. The mining algorithm is applied to training data and recursively partition the dataset until each subset contains one class or no further test is available. The result is often a complex tree that overfits the data. The overfit problem reduces the accuracy when applied to unseen data. The pruning of the decision tree is the process of removing leaves and branches to improve the accuracy and performance of the decision tree. Typically, the tree pruning methods use statistical measures to remove the least reliable sub-trees and consequently, result in faster classification and an improvement the accuracy of the tree.

The pruning of the decision tree is established by replacing the undesired sub tree by a leaf node. The replacement takes place if the expected error rate of the sub tree is greater than in the leaf node [31]. Getting a minimal decision tree is considered to be very important than selecting good split in terms of quality of decision tree [9].

The following subsections introduce the commonly used pruning techniques of tree induction algorithms, pre-pruning and Post-pruning.

5.1 Pre-Pruning Strategies

In pre-pruning approach, a tree is pruned by stopping its construction by deciding not to further partition the subset of training data at a given node. As a consequence, a node becomes a leaf that holds a class value with the most frequent class among the subset of samples or simply the probability distribution of those samples. Pre-pruning criteria are based on *statistical significance* [17], *information gain* [18], or *error reduction* [15,20]. For instance, a mining algorithm may determine either to stop or grow the tree at a given node by setting the minimum gain to 0.01 and further data partitioning is prevented if the computed information gain at each node less that this

threshold value. This approach is adopted by CHAID decision tree based classifier.

The approaches presented in [11,32] push the accuracy and size constraints into the decision tree in order to prune the tree dynamically. They proposed PUBLIC classifier that integrates building and pruning in one stage. In PUBLIC, a node in not further expanded in the construction stage of decision tree if it is determined that it is certain to be pruned in the subsequent pruning stage.

5.2 Post-Pruning of Decision Trees

Post-pruning removes branches from the completely grown tree, by traversing the constructed tree and uses the estimated errors to decide whether some undesired branches should be replaced by a leaf node or not [7,21]. This replacement is the key issue of many pruning criteria that appear in the literature.

There are two ways of post-pruning techniques that have been studied in data mining algorithms. They are basically based on whether to use the same training dataset that has been used for construction the decision tree or to use a test set that is not used in training tree models. The key issue and the major difficulty to the first approach are to derive an accurate estimate of the error rate when the trained model is used to classify previously unseen data. That is not an issue in the second approach, which reserves some of the date for testing, therefore, the model has to be built based on a smaller training dataset. A common solution to this problem is to use cross-validation procedure. In a 10 fold cross-validation procedure, the entire dataset is first randomly divided into 10 equal sized blocks. Then, a tree model is constructed using 90% of the data (training set) and testing the remaining 10% (testing set). Next, another tree is constructed, but based on different training and testing data. This process is repeated 10 times using different training and testing sets. The final tree size and estimated error is the average size and error of the ten optimally pruned trees. One disadvantage of this procedure is that, it is computationally expensive.

Several pruning techniques have been proposed based on *cost-complexity* [9,21], *reduced-error* [18,21], *pessimistic-error* [7,18,21], *minimum-error, critical value* [21] and *Minimum Description Length (MDL)* [22]. The objective of such criteria is to find simple and comprehensible tree with acceptable accuracy.

Empirical evaluation has shown that, post-pruning approach is more effective than pre-pruning [7,9,21]. This primarily because pre-pruning methods are based essentially on heuristic rules while post-pruning methods are based on statistical theories. Many decision tree algorithms, however, incorporate both approaches but primarily rely on post-pruning to obtain optimal decision tree. Since most of the pre-pruning methods are based on heuristic rules and considered very simple, while post-pruning methods are more complex and the most popular approaches, the following section discuss the methods of post-pruning of decision tree in great details.

5.2.1 Reduced Error Pruning (REP)

This method proposed by [18] and involves the use of a test dataset directly in the process of constructed pruned trees, rather than to be used only for determining the best tree, as in cost-complexity pruning. Because the procedure does not require building a sequence of sub trees, it is claimed to be faster than cost-complexity pruning.

The method works by beginning with using the test data on the unpruned tree and record the number of cases corresponding to each class in each node. Then, for each internal node, count the number of test errors if the branch rooted at this node is kept and the number if it is pruned to a leaf. The difference between them is a measure of the gain (if positive) or loss (if negative) of pruning the branch. Next, select the node with the largest gain and prune its branch off. This gives the first pruned sub tree. Applying same procedure repeatedly to the previously pruned tree will result in obtaining a shrinking tree. The problem that may arise using REP is that, at a certain point, further pruning may cause increasing in test errors. In such case, the process stops at this point and the last and smallest sub tree is declared the final pruned tree.

A major advantage of REP lies in its linear computational complexity, since to evaluate the chance of pruning; each node is to be visited only once. On the other hand, its disadvantage arises in its bias toward over pruning due to the fact that all evidence encapsulated in the training set is neglected during the pruning process.

5.2.2 Cost-Complexity Pruning (CCP)

This method is also known as the CART pruning algorithm. It has been introduced by [9] and implemented in CART, S-Plus. CCP uses the train/test approach for pruning, which trains the model on one set and tests it on another. Since CCP is somewhat complicated procedure, we will explain it through the following example. In our discussion we use the notion of a sub tree to indicate a pruned tree that has the same root of the unpruned tree and a notion of a branch to indicate a segment of tree that can be a candidate for pruning.

Given a branch T, the cost complexity measure of T

rooted at a node *t* is defined as:

$$R_\alpha(T_t) = R(T_t) + \alpha N_T \longrightarrow (3.1)$$

Where $R(T_t)$, called cost, is the error rate calculated by dividing the total number of error cases in all leaves of branch *T* by the total number of cases in the entire dataset; and N_t, called complexity, is the number of leaves in *T*. The parameter α, which is non-negative by definition, can be interpreted as the cost per extra leaf. In figure 1, for example, we have a branch rooted at node 40, whose complexity is 3 (the number of leaves). The entire dataset contains 500 cases with 3 classes, whose distribution in this branch is shown on the second line of each node. If branch *T* is not pruned, then the cost is calculated as:

$$R(T_{40}) = ((8+0) + (10+0) + (0+0))/500 = 18/500$$

If the branch is pruned, the node *t* becomes a leaf of class 1, and its cost is computed as:

$$R(40) = (20+1)/500 = 21/500$$

The cost complexity measure, when the branch is pruned to a leaf is given by

$$R_\propto(t) = R(t) + \propto \longrightarrow (3.2)$$

Where $\propto$ is sufficiently small, $R_\propto(t)$ is greater than $R_\propto(T_t)$, since $R(t)$ is always greater than $R(T_t)$. When the value of $\propto$ Increases to exceed a critical value, $R_\propto(T_t)$ becomes greater than $R_\propto(t)$, because the complexity terms $\propto N_T$ will dominate. Then, pruning of T is preferable since its cost-complexity is smaller. To find this critical value of α, equate (3.1) and (3.2), and then solve for α. We have

$$\propto = (R(t) - R(T_t))/(N_T - 1) \longrightarrow (3.3)$$

Hence, for branch T_{40},

$$\propto = (21/500 - 18/500)/3\text{-}1 = 0.003$$

Similarly, for branch T_{41}:

$$\propto = (20/500 - 18/500)/(2\text{-}1) = 0.004$$

The CCP works as follows: the algorithm starts by calculating the value of $\propto$ for each branch, rooted at each different internal node of the unpruned tree. The branch that has the smallest value of $\propto$ is then pruned, yielding the first pruned sub tree. If several $\propto$'s are tied, as the smallest, then corresponding branches are all pruned away. Next, the values of $\propto$ are computed again, but based on the last pruned tree; this will prune away another branch. Repeating this process will progressively produce a series of smaller sub trees; each nested within the previous one. Each sub tree produced in this procedure is optimal with respect to size; that is, no other sub tree of the same size would have lower error rate than the one obtained by this procedure. After the series of sub trees are generated, each of them is used to classify a test dataset. Ideally, the final pruned tree would be the one that has the lowest test error rate.

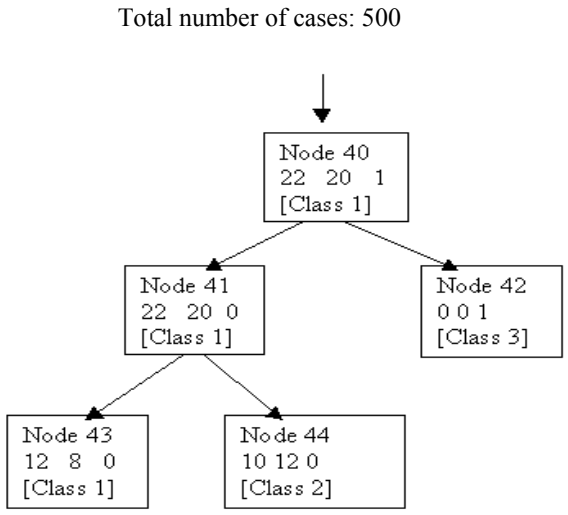

Figure 1. An example of cost –complexity pruning.

In the above example, suppose the $\propto$ value of the other branches (not shown in the diagram) is greater than 0.003, then branch T_{40} is selected to be pruned the first. Notice that branch T_{41} will never be selected. This implies that the sequence of pruned trees generated by this method does not necessarily have its size decreased by one leaf each time.

5.2.3 Pessimistic-Error Pruning (PEP)

This method was proposed by [7,18], and has been implemented in C4.5. The method uses training dataset rather than using testing dataset and stands on more solid statistical methods.

Suppose there are *n* training cases in a leaf, *e* of them misclassified. C4.5 deals with this set of data as a sample drawn from binomial population, i.e. observing *e* events in *n* trials (This is in fact, is not the case, as Quinlan pointed out). Then, the method tries to estimate the population parameter, which is the error rate on unseen data, based on the information contained in this sample. The method pessimistically uses the upper confidence bound of the binomial distribution, denoted by $U_\alpha(e,n)$, as the estimated error rate at this leaf. So, a leaf covering *m* training cases with an estimated error rate of $U_\alpha(e,n)$ would be expected to have $mU_\alpha(e,n)$ error cases. Similarly, the estimated number of errors for a branch is just the sum of the estimated errors of its sub-branches. If the estimated number of errors for a branch is greater than or equal to the number when it is regarded as a leaf, the branch is pruned; otherwise, the branch is maintained.

To understand how this method works, let us look again to the example shown in Figure 1. First, we determine a

confidence level of 90% (the default confidence value; in C4.5 is 75% which often produce a pruned tree that is too large). For node 43 and node 44, which are leaves, we have $U_{0.1}$ (7,20) = 0.5673 and $U_{0.1}$ (10,22) = 0.6112, respectively. Now, the estimated number of errors for branch T_{41} is:20(0.5673) + 22 (0.6112) = 24.793

If the branch is pruned to a leaf, the estimated number of errors is

$$42U_{0.1}\ (20,42) = 42\ (0.5858) = 24.604$$

In this case, branch T_{41} should be pruned since this would reduce the estimated number of errors. After pruning, the estimated number of errors for branch T_{40} is $42U_{0.1}$ (20,22) + 1 $U_{0.1}$ (0,1) = 42 (0.5858) + 1 (0.9) = 25.504

If it is pruned, the estimated number of errors will be
43 $U_{0.1}$ (21,43) = 43 (0.5962) = 25.638
Pruning of branch T_{40} would cause increasing the estimated number of errors, so it is retained (with two leaves, node 41 and node 42) and included in the final pruned tree.

5.2.4 Comparison of Pruning Techniques

As we have seen earlier, different pruning methods would lead to different results. Many empirical studies have been conducted to evaluate the effectiveness of various pruning methods [8,18,21]. It has been shown that, no single method is best of the others. In terms of classification accuracy, the cost-complexity and reduced-error methods appear to perform somewhat better in many domains. However, these pruning methods normally run more slowly than those that depend on the testing dataset.

6. Conclusion

The goal of this paper is to provide a comprehensive survey about classical and incremental classification algorithms. We focus our attention on decision tree based classifiers and its applications to solve data mining problems. Many important issues that distinguish between each classifier such as splitting criteria and pruning methods were discussed. Such criteria lead to the variation of decision tree based classification.

References

[1] Han, J., and Kamber, M., "Data Mining: Concepts and Techniques", 1st Edition, Harcourt India Private Limited. 2001.

[2] Duda, R. O., Hart, P. E. and Stork, D. G., "Pattern Classification", 2nd Edition, John Wiley & Sons (Asia) PV. Ltd., 2002.

[3] Patterson, D. W., "Introduction to Artificial Intelligence and Expert Systems", 8th Edition, Prentice-Hall, India, 2000.

[4] Berry Michael J. A. and Linoff Gorden S.,"Mastering Data Mining", John Wiley & Sons, 2000.

[5] Robert Groth , " Data Mining: A hands-On Approach for business professionals", Prentice Hall PTR, New Jersey, USA, 1998.

[6] Quinlan, J. R., "Induction of Decision Trees", Machine Learning, 1:1, Boston: Kluwer, Academic Publishers, 1986, 81-106.

[7] Quinlan, J. R., "C4.5: Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann, 1993.

[8] Esposito, F., Malerba, D. and Semeraro, G., "A Comparative Analysis of Methods for Pruning Decision Trees", IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Computer Society, 1997.

[9] Breiman, L. J., Friedman, J. H., Olshen, R. A. and Stone, C. J. , Classifications and Regression Trees, New York, Chapman and Hall., 1984.

[10] Shafer, J., Aggrawal, R. and Mehta, M., "SPRINT: A Scalable Parallel Classifier for Data Mining", In Proceedings of 22nd VLDB Conference, 1996.

[11] Rastogi, R. and Shim, K.., "PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning", In Proceedings of the 24th International Conference on VLDB, 1998.

[12] Gehrke, J., Ganti, V, Ramakrishnan, R. and Loh, W-Y., "BOAT-optimistic Decision Tree Construction", In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999.

[13] Mitchell, T. M, Utgoff, P. E. and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics", In machine learning: Artificial Intelligence Approach, Edited by R. S. Michalski, J. G. Carbonell, and Michall T. M., Tioga publishing Co., Palo Alto, CA, USA, 1983.

[14] Heath, D., Kasif, S., and Salzburg, S., "Learning Oblique Decision Trees", In Proceedings of the 13th International Joint Conference on Artificial Intelligence, San Mateo, CA: Morgan Kaufmann, 1993.

[15] Liu, W. Z. and White, A. P., "The Importance of Attribute Selection Measures in Decision Tree Induction", Machine Learning, 15, 1994.

[16] Fayyad, U. M. and Irani, K.B., "The Attribute Selection Problem in Decision Tree Generation", In Proceedings of 10th National Conference on Artificial Intelligence, Menlo Park, CA: AAA Press/MIT Press, 1992.

[17] Clark, P. and Niblett, T., "The CN2 Induction Algorithm", Machine Learning, 3(4), 1989.

[18] Quinlan J. R., "Simplifying Decision Trees", International Journal of Machine Learning Studies, 27, 1987, 221-234.

[19] Clark, L.A and pregiban, D., "Tree-Based Models", In statistical Models in (J.M. Chambers and T.J. Hastie eds.), pacific Grave, CA: Wadsworth and Brooks, 1992.

[20] Kass, G. V., "An Exploratory Technique for Investigating Large quantities of Categorical Data", Applied Statistics, 29,1980, 119-127.

[21] Mingers, J., "An Empirical Comparison of Pruning Methods for Decision Tree Induction", Machine Learning, 4(2), 1987.

[22] Rissanen, J., "Stochastic Complexity in Statistical Inquiry", World Scientific Publicarion Co., 1989.

[23] Kalles, D. and Morris, T., "Efficient Incremental Induction of Decision Trees", Machine Learning, 24, 1996, pp. 231-241.

[24] Utgoff, P. E., "ID5: An Incremental ID3", In Proceedings of the 5th International Conference on Machine Learning, 1988, pp. 107-120.

[25] Utgoff, P. E., "Incremental Induction of Decision Tress", Machine Learning, 4(2), 1989, pp.161-186.

[26] Fisher, D., "Knowledge Acquisition via Incremental Conceptual Clustering", Machine Learning, 2, 1987, pp.139-172.

[27] Schlimmer, J. C. and Fisher, D., "A Case Study of Incremental Concept Induction", In Proceedings of the 5th National Conference on Artificial Intelligence, 1986, pp.496-501, Philadelphia, PA, Morgan Kaufman.

[28] Van-de-Velde, W. ,"The Incremental Induction of Topologically Minimal Decision Trees" In Proceedings of 7th International Conference on Machine Learning, Austin, TX., 1990, 66-74.

[29] Crawford, S., "Extensions to the CART Algorithm", International Journal of Man-Machine Studies 31(2), 1989, 197-217.
[30] Cockett, J. and Zhu, Y., "A New Incremental Technique for Decision Trees with Thresholds" In Proceedings of the SPIE 1095, 1989, 804-811.
[31] Pujari, A. K., "Data Mining Techniques", 1st Edition, Universities Press (India) Limited, 2001.
[32] Garofalakis, M., Hyun, D., Rastogi, R. and Shim, K., "Efficient Algorithms for Constructing Decision Trees with Constraints", Bell Laboratories Tech. Memorandum, 2000.
[33] Clair, St. C., "A Usefulness Metric and its Application to Decision Tree Based Classification", Ph.D. Thesis, School of Computer Science, Telecommunications and Information Systems, DePaul University, Chicago, USA, 1999.
[34] Mingers, J., "An Empirical Comparison of Selection Measures for Decision Tree Induction", Machine learning, 3, 1989.

Ahmed Sultan Al-Hegami received His B.Sc degree in Computer Science from King Abdul Aziz University, Saudi Arabia, MCA (Master of Computer Application) from Jawaharlal Nehru University, New Delhi, India; and Ph.D. degree from University of Delhi, Delhi, India. He is lecturer at the Department of Computer Science, Sana'a University, Yemen.

Currently he is assistant professor at the Department of Computer Science, Sana'a University, Yemen. His research interest includes artificial intelligence, machine learning, temporal databases, real time systems, data mining, and knowledge discovery in databases.