UNIT IV

Classification: Decision Tree Induction, Bayesian Classification, Rule Based Classification, Classification by Back Propagation, Support Vector Machines, Lazy Learners, Model Evaluation and Selection, Techniques to improve Classification Accuracy

Classification is a form of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical (discrete, unordered) class labels. For example, we can build a classification model to categorize bank loan applications as either safe or risky. Such analysis can help provide us with a better understanding of the data at large. Many classification methods have been proposed by researchers in machine learning, pattern recognition, and statistics.

Why Classification?

A bank loans officer needs analysis of her data to learn which loan applicants are "safe" and which are "risky" for the bank. A marketing manager at AllElectronics needs data analysis to help guess whether a customer with a given profile will buy a new computer.

A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive. In each of these examples, the data analysis task is classification, where a model or classifier is constructed to predict class (categorical) labels, such as "safe" or "risky" for the loan application data; "yes" or "no" for the marketing data; or "treatment A," "treatment B," or "treatment C" for the medical data.

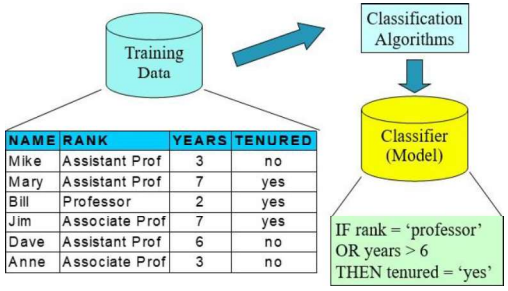
Suppose that the marketing manager wants to predict how much a given customer will spend during a sale at AllElectronics. This data analysis task is an example of **numeric prediction**, where the model constructed predicts a continuous-valued function, or ordered value, as opposed to a class label. This model is a **predictor**.

Regression analysis is a statistical methodology that is most often used for numeric prediction; hence the two terms tend to be used synonymously, although other methods for numeric prediction exist. Classification and numeric prediction are the two major types of **prediction problems**.

General Approach for Classification:

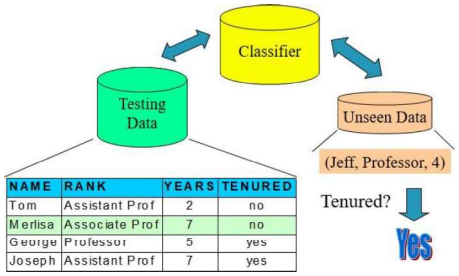
Data classification is a two-step process, consisting of a *learning step* (where a classification model is constructed) and a *classification step* (where the model is used to predict class labels for given data).

- In the first step, a classifier is built describing a 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 database tuples and their associated class labels.
- Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
- In the second step, the model is used for **classification**. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic, because the classifier tends to overfit the data.
- Accuracy rate is the percentage of test set samples that are correctly classified by the model



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Fig: Learning Step



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Fig: Classification Step

Decision Tree Induction:

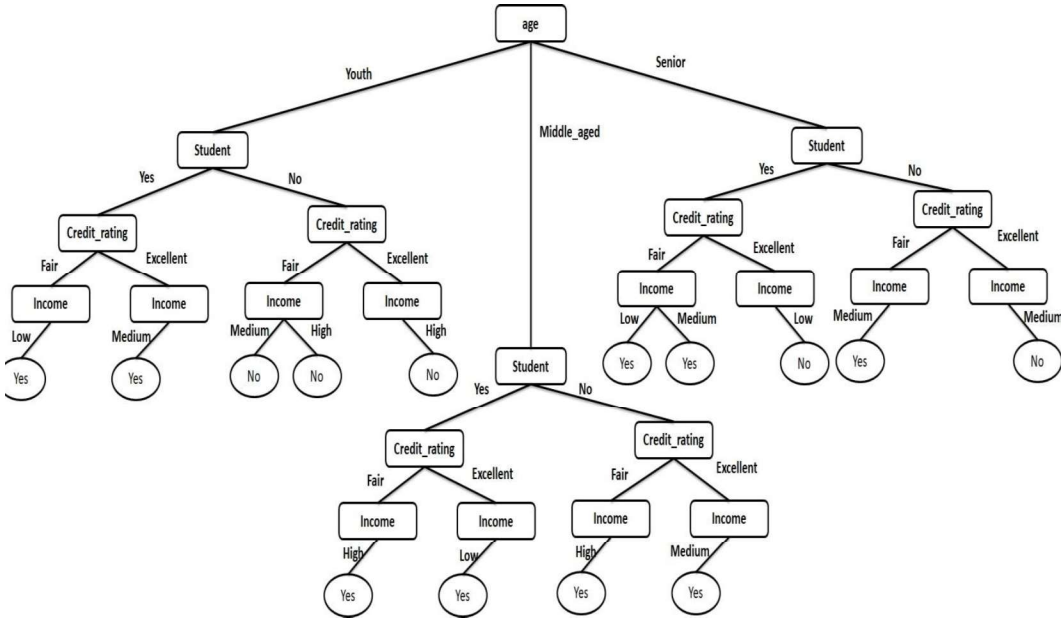
Decision tree induction is the learning of decision trees from class-labeled training tuples. A **decision tree** is a flowchart-like tree structure, where each **internal node** (non leaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label. The topmost node in a tree is the **root** node. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.

"How are decision trees used for classification?" Given a tuple, ***X***, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

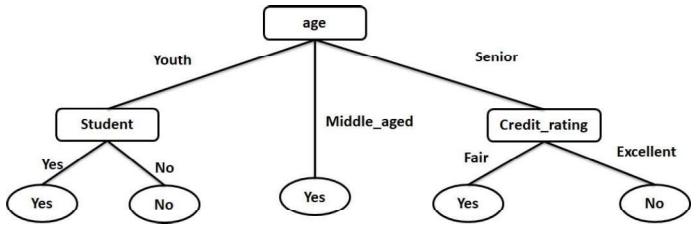
"Why are decision tree classifiers so popular?" The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast.

Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

The Tree after splitting branches is



The Tree after Tree Pruning,



Finally, The Classification Rules are,

- **IF** age=Youth **AND** Student=Yes **THEN** buys_computer=Yes
- **IF** age=Middle_aged **THEN** buys_computer=Yes
- **IF** age=Senior **AND** Credit_rating=Fair **THEN** buys_computer=Yes

Age	P	N	TOTAL	I(P,N)	
youth	2	3	5	I(2,3	0.970
middle_aged	4	0	4	I(4,0)	0
senior	3	2	5	I(3,2)	0.970

$$Info_{Age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.693$$

Gain(Age) = Info(D) – $Info_{Age}(D)$

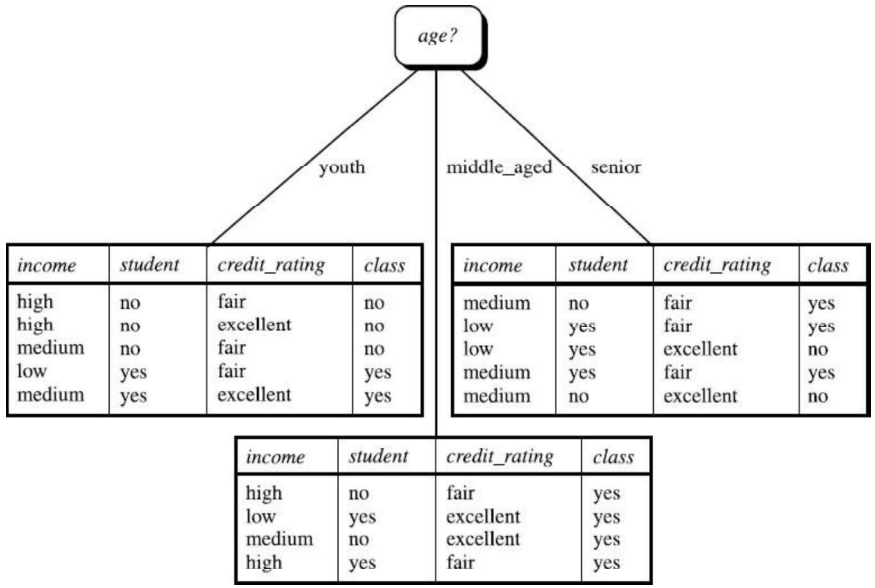
= 0.940 – 0693 = 0.247

Similarly,

Gain(Income) = 0.029 Gain (Student) = 0.151

Gain (credit_rating) = 0.048

Finally, *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node *N* is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

During tree construction, *attribute selection measures* are used to select the attribute that best partitions the tuples into distinct classes. When decision trees are built, many of the branches may reflect noise or outliers in the training data. *Tree pruning* attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

- During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as **ID3** (Iterative Dichotomiser).
- This work expanded on earlier work on *concept learning systems*, described by E. B. Hunt, J. Marin, and P. T. Stone. Quinlan later presented **C4.5 (a successor of ID3)**, which became a benchmark to which newer supervised learning algorithms are often compared.
- In 1984,a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book *Classification and Regression Trees* (**CART**), which described the generation of binary decision trees.

Decision Tree Algorithm:

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute list*, the set of candidate attributes;
- *Attribute selection method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

1) create a node *N*;
2) **if** tuples in *D* are all of the same class, *C*, **then**
3) return *N* as a leaf node labeled with the class *C*;
4) **if** *attribute list* is empty **then**
5) return *N* as a leaf node labeled with the majority class in *D*; // majority voting
6) apply **Attribute selection method**(*D*, *attribute list*) to **find** the "best" *splitting criterion*;
7) label node *N* with *splitting criterion*;
8) **if** *splitting attribute* is discrete-valued **and**

multiway splits allowed **then** // not restricted to binary trees

9) *attribute list* ←*attribute list - splitting attribute*; // remove *splitting attribute*
10) **for each** outcome *j* of *splitting criterion*

// partition the tuples and grow subtrees for each partition

11) let *Dj* be the set of data tuples in *D* satisfying outcome *j*; // a partition
12) **if** *Dj* is empty **then**
13) attach a leaf labeled with the majority class in *D* to node *N*;
14) **else** attach the node returned by **Generate decision tree**(*Dj* , *attribute list*) to node *N*;

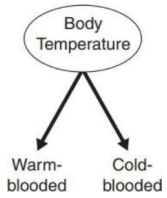
endfor

15) return *N*;

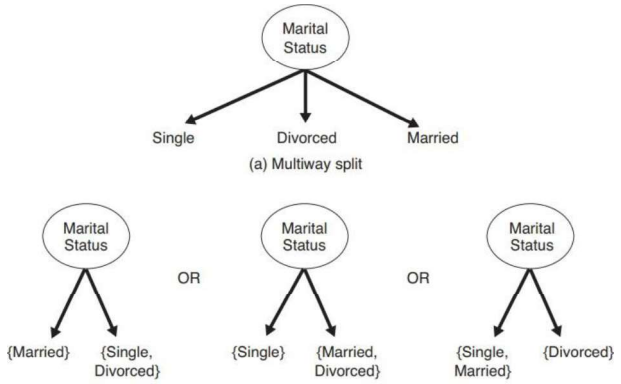
Methods for selecting best test conditions

Decision tree induction algorithms must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.

Binary Attributes: The test condition for a binary attribute generates two potential outcomes.

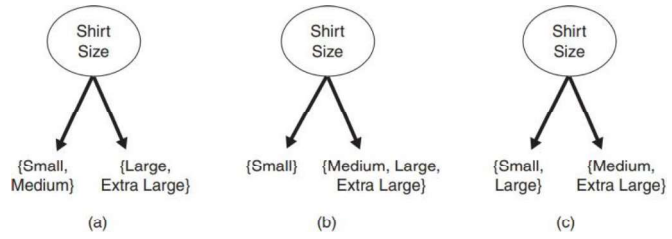


Nominal Attributes: These can have many values. These can be represented in two ways.



(b) Binary split {by grouping attribute values}

Ordinal attributes: These can produce binary or multiway splits. The values can be grouped as long as the grouping does not violate the order property of attribute values.



Example for Decision Tree construction and Classification Rules:

Construct Decision Tree for following dataset,

Age	income	Student	credit_rating	buys_computer
youth	high	No	fair	No
youth	high	No	excellent	No
middle_aged	high	No	fair	Yes
senior	medium	No	fair	Yes
senior	low	Yes	fair	Yes
senior	low	Yes	excellent	No
middle_aged	low	Yes	excellent	Yes
youth	medium	No	fair	No
youth	low	Yes	fair	Yes
senior	medium	Yes	fair	Yes
youth	medium	Yes	excellent	Yes
middle_aged	medium	No	excellent	Yes
middle_aged	high	Yes	fair	Yes
senior	medium	No	excellent	No

Solution:

Here the target class is buys_computer and values are yes, no. By using ID3 algorithm, we are constructing decision tree.

For ID3 Algorithm we have calculate Information gain attribute selection measure.

CLASS	P	buys_computer (yes)	9
	N	buys_computer (no)	5
		TOTAL	14

$$\text{Info}(D) = I(9,5) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

Age	P	N	TOTAL	I(P,N)
youth	2	3	5	I(2,3
middle_aged	4	0	4	I(4,0)
senior	3	2	5	I(3,2)

$$I(2,3) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.970$$

$$I(4,0) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0$$

$$I(3,2) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.970$$

- This set is independent of the training set used to build the unpruned tree and of any test set usedfor accuracy estimation.
- The algorithm generates a set of progressively pruned trees. Ingeneral, the smallest decision tree that minimizes the cost complexity is preferred.
- C4.5 uses a method called **pessimistic pruning**, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning.

Scalability of Decision Tree Induction:

"What if D, the disk-resident training set of class-labeled tuples, does not fit in memory? In other words, how scalable is decision tree induction?" The efficiency of existing decision tree algorithms, such as ID3, C4.5, and CART, has been well established for relatively small data sets. Efficiency becomes an issue of concern when these algorithms are applied to the mining of very large real-world databases. The pioneering decision tree algorithms that we have discussed so far have the restriction that the training tuples should reside *in memory*.

In data mining applications, very large training sets of millions of tuples are common. Most often, the training data will not fit in memory! Therefore, decision tree construction becomes inefficient due to swapping of the training tuples in and out of main and cache memories. More scalable approaches, capable of handling training data that are too large to fit in memory, are required. Earlier strategies to "save space" included discretizing continuous-valued attributes and sampling data at each node. These techniques, however, still assume that the training set can fit in memory.

Several scalable decision tree induction methods have been introduced in recent studies. Rain Forest, for example, adapts to the amount of main memory available and applies to any decision tree induction algorithm. The method maintains an **AVC-set** (where "AVC" stands for "*Attribute-Value*, *Classlabel*") for each attribute, at each tree node, describing the training tuples at the node. The AVC-set of an attribute A at node N gives the class label counts for each value of A for the tuples at N. The set of all AVC-sets at a node N is the **AVC-group** of N. The size of an AVC-set for attribute A at node N depends only on the number of distinct values of A and the number of classes in the set of tuples at N. Typically, this size should fit in memory, even for real-world data. Rain Forest also has techniques, however, for handling the case where the AVC-group does not fit in memory. Therefore, the method has high scalability for decision tree induction in very large data sets.

age	*buys_computer* yes	no
youth	2	3
middle_aged	4	0
senior	3	2

income	*buys_computer* yes	no
low	3	1
medium	4	2
high	2	2

student	*buys_computer* yes	no
yes	6	1
no	3	4

credit_ratting	*buys_computer* yes	no
fair	6	2
excellent	3	3

Fig: AVC Sets for dataset

Attribute Selection Measures

- An **attribute selection measure** is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes.
- If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class).
- Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as **splitting rules** because they determine how the tuples at a given node are to be split.
- The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure4 is chosen as the splitting attribute for the given tuples.
- If the splitting attribute is continuous-valued or if we are restricted to binary trees, then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion.
- The tree node created for partition D is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly.
- There are three popular attribute selection measures—*information gain, gain ratio*, and *Gini index*.

Information Gain

ID3 uses **information gain** as its attribute selection measure. Let node N represent or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

Where $p_i$is the nonzero probability that an arbitrary tuple in D belongs to class $C_i$and is estimated by $|C_i,D|/|D|$. A log function to the base 2 is used, because the information is encoded in bits.$Info(D)$ is also known as the **entropy** of D.

Information needed after using A to split D into V partitions.

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D).$$

The attribute A with the highest information gain, $Gain(A)$, is chosen as the splittingattribute at node$N$. This is equivalent to saying that we want to partition on the attribute$A$ that would do the "best classification," so that the amount of information still requiredto finish classifying the tuples is minimal.

Gain Ratio

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D) as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

This value represents the potential information generated by splitting the trainingdata set, D, into v partitions, corresponding to the v outcomes of a test on attribute A. Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in D. It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}.$$

Gini Index

The Gini index is used in CART. Using the notation previously described, the Gini indexmeasures the impurity of D, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

Where $p_i$is the nonzero probability that an arbitrary tuple in D belongs to class $C_i$and is estimated by $|C_i,D|/|D|$ over m classes.

Note: The Gini index considers a binary split for each attribute.

When considering a binary split, we compute a weighted sum of the impurity of eachresulting partition. For example, if a binary split on A partitions D into D_1 and D_2, the Gini index of D given that partitioning is

$$Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2).$$

- For each attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.
- For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point.
- The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

Tree Pruning:

- When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers.
- Tree pruning methods address this problem of *overfitting* the data. Such methods typically use statistical measures to remove the least-reliable branches.
- Pruned trees tend to be smaller and less complex and, thus, easier to comprehend.
- They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

"How does tree pruning work?" There are two common approaches to tree pruning:

prepruning and *postpruning*.

- In the **prepruning** approach, a tree is "pruned" by halting its construction early. Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.
- If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. There are difficulties, however, in choosing an appropriate threshold.
- In the postpruning, which removes subtrees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.

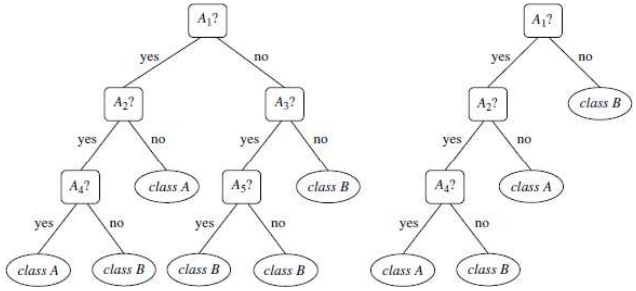


Fig: Unpruned and Pruned Trees

- The **cost complexity** pruning algorithm used in CART is an example of the postpruning approach.
- This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the **error rate** is the percentage of tuples misclassified by the tree). It starts from the bottom of the tree.
- For each internal node, N, it computes the cost complexity of the subtree at N, and the cost complexity of the subtree at N if it were to be pruned (i.e., replaced by a leaf node).
- The two values are compared. If pruning the subtree at node N would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept.
- A **pruning set** of class-labeled tuples is used to estimate cost complexity.