at Is Classification?

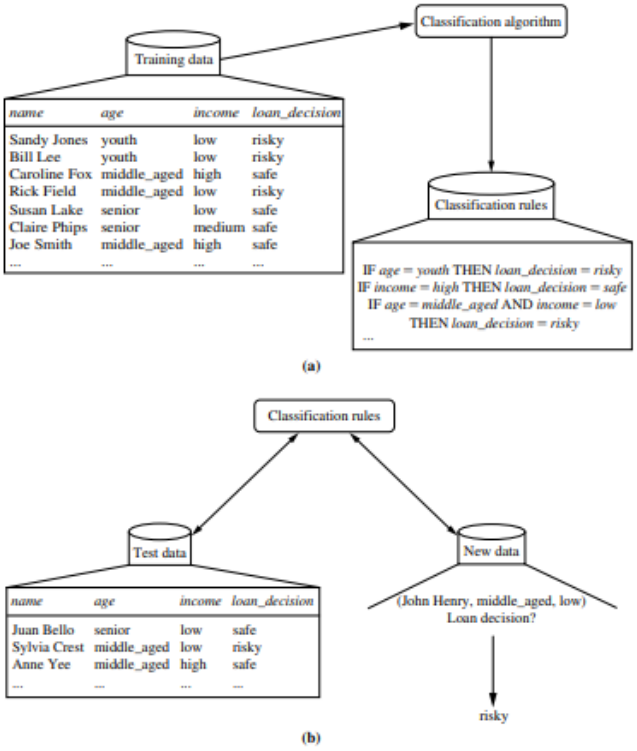
A bank loans officer needs analysis of her data to learn which loan applicants are “safe” and which are “risky” for the bank. A marketing manager at AllElectronics needs data analysis to help guess whether a customer with a given profile will buy a new computer

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.

General Approach to Classification \

“How does classification work?”

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). The process is shown for the loan application data of Figure 8.1. (The data are simplified for illustrative purposes. In reality, we may expect many more attributes to be considered. 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. A tuple, X, is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2,..., An. 1 Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute. The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples and are randomly sampled from the database under analysis. In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects. 2



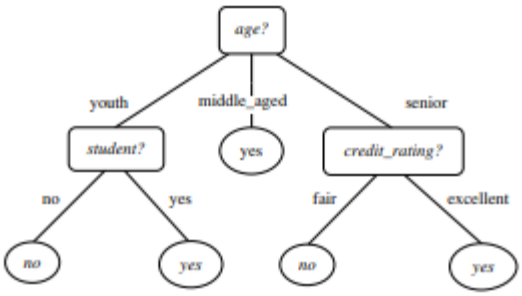
What about classification accuracy?"

In the second step (Figure 8.1b), 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 (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.The accuracy of a classifier on a given test set is the percentage of 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

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 (nonleaf 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. A typical decision tree is shown in Figure 8.2. It represents the concept buys computer, that is, it predicts whether a customer at AllElectronics is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce only

binary trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.



“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 Algorithm

The algorithm is called with three parameters: D, attribute list, and Attribute selection method. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter attribute list is a list of attributes describing the tuples. Attribute selection method specifies a heuristic procedure for selecting the attribute that “best” discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

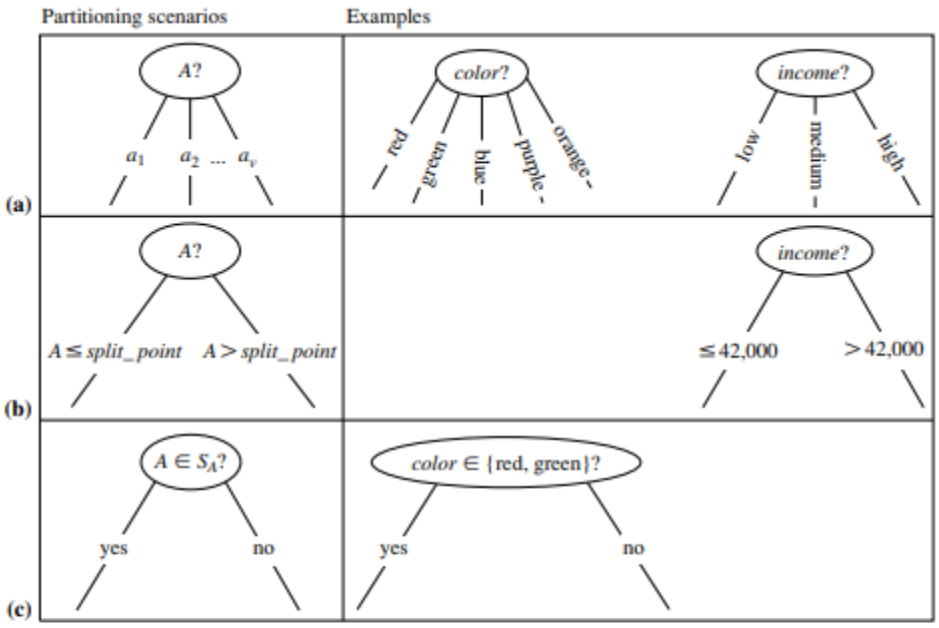
Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

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.

The algorithm calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the "best" way to separate or partition the tuples in D into individual classes. The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. T



This figure shows three possibilities for partitioning tuples based on the splitting criterion, each with examples. Let *A* be the splitting attribute. (a) If *A* is discrete-valued, then one branch is grown for each known value of *A*. (b) If *A* is continuous-valued, then two branches are grown, corresponding to $A \leq split\_point$ and $A > split\_point$. (c) If *A* is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for *A*.

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. e 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

Three popular attribute selection measures—information gain, gain ratio, and Gini index.

Information Gain: 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

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

Info(D) is also known as the entropy

How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

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

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,

$$\text{Gain(A)} = \text{Info(D)} - \text{InfoA (D)}$$

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Induction of a decision tree using information gain

Let class C1 correspond to yes and class C2 correspond to no. There are nine tuples of class yes and five tuples of class no. A (root) node N is created for the tuples in D. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Eq. (8.1) to compute the expected information needed to classify a tuple in D:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute age. We need to look at the distribution of yes and no tuples for each category of age. For the age category "youth," there are two yes tuples and three no tuples. For the category "middle aged," there are four yes tuples and zero no tuples

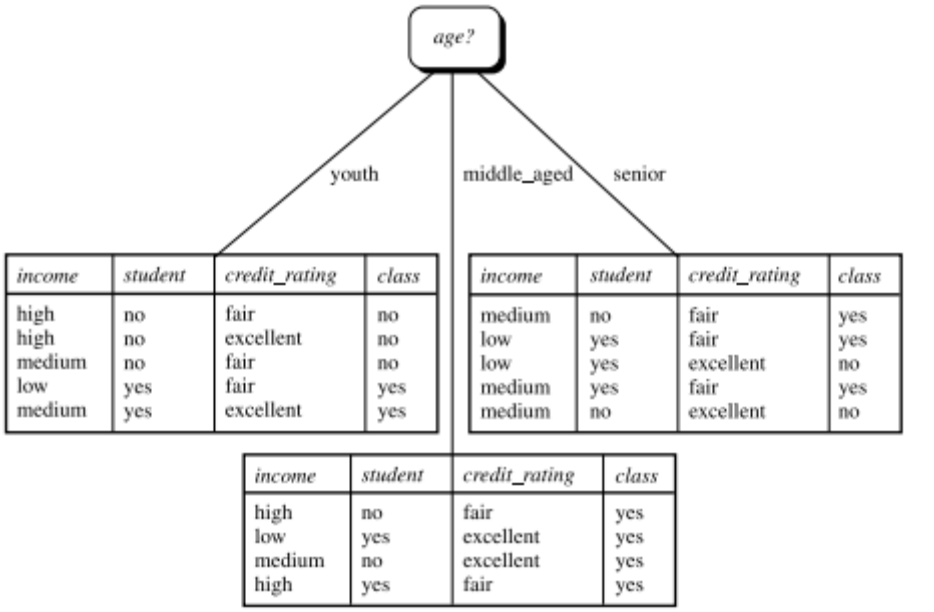
$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right)$$
$$+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$
$$= 0.694 \text{ bits.}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute Gain(income) = 0.029 bits, Gain(student) = 0.151 bits, and Gain(credit rating) = 0.048 bits. Because 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 shown in Figure



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

The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

Gain Ratio

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier such as product ID. A split on product ID would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be Infoproduct ID(D) = 0. Therefore, the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification

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 training data 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

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

The attribute with the maximum gain ratio is selected as the splitting attribute. Note, however, that as the split information approaches 0, the ratio becomes unstable. A constraint is added to avoid this, whereby the information gain of the test selected must be large—at least as great as the average gain over all tests examined.

Computation of gain ratio for the attribute *income*. A test on *income* splits the data of Table 8.1 into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*, we first use Eq. (8.5) to obtain

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right)$$

$$= 1.557.$$

From Example 8.1, we have $Gain(income) = 0.029$. Therefore, $GainRatio(income) = 0.029/1.557 = 0.019$. ■

Gini Index

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

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

where pi is the probability that a tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|. The sum is computed over m classes

Induction of a decision tree using the Gini index. Let D be the training data shown earlier in Table 8.1, where there are nine tuples belonging to the class *buys_computer* = *yes* and the remaining five tuples belong to the class *buys_computer* = *no*. A (root) node N is created for the tuples in D. We first use Eq. (8.7) for the Gini index to compute the impurity of D:

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

To find the splitting criterion for the tuples in D, we need to compute the Gini index for each attribute. Let's start with the attribute income and consider each of the possible splitting subsets. Consider the subset {low, medium}. This would result in 10 tuples in partition D1 satisfying the condition "income ∈ {low, medium}." The remaining four tuples of D would be assigned to partition D2. The Gini index value computed based on

$$\begin{aligned} &Gini_{income \in \{low,medium\}}(D) \\ &\quad = \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\ &\quad = \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\ &\quad = 0.443 \\ &\quad = Gini_{income \in \{high\}}(D). \end{aligned}$$

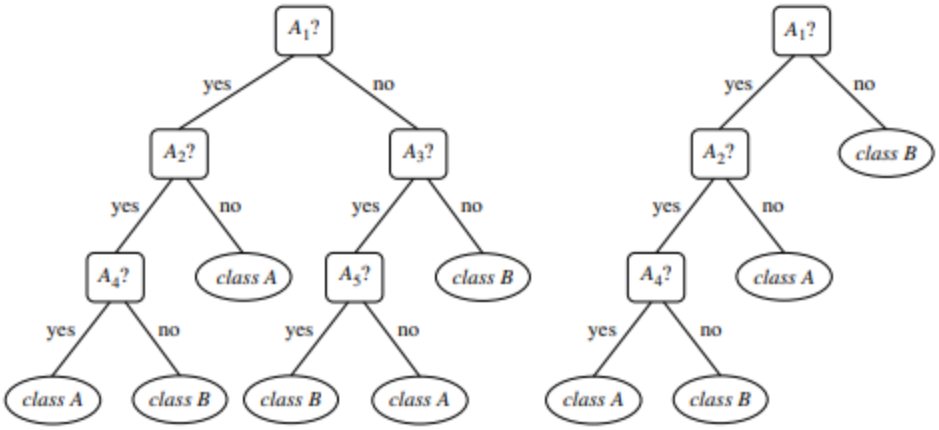
Similarly, the Gini index values for splits on the remaining subsets are 0.458 (for the subsets {low, high} and {medium}) and 0.450 (for the subsets {medium, high} and {low}). Therefore, the best binary split for attribute income is on {low, medium} (or {high}) because it minimizes the Gini index. Evaluating age, we obtain {youth, senior} (or {middle aged}) as the best split for age with a Gini index of 0.375; the attributes student and credit rating are both binary, with Gini index values of 0.367 and 0.429, respectively. The attribute age and splitting subset {youth, senior} therefore give the minimum Gini index overall, with a reduction in impurity of 0.459 − 0.357 = 0.102. The binary split "age ∈ {youth, senior?}" results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion. Node N is labeled with the criterion, two branches are grown from it, and the tuples are partitioned accordingly.

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. An unpruned tree and a pruned version of it are shown in Figure 8.6. 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 (e.g., by deciding not to further split or partition the subset of training tuples at a given node). 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

The second and more common approach is 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. For example, notice the subtree at node "A3?" in the unpruned



An unpruned decision tree and a pruned version of it.

Suppose that the most common class within this subtree is "class B." In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "class B.