Classification: Basic concepts, general approach to solving a classification problem, decision tree induction: working of decision tree, building a decision tree, methods for expressing attribute test conditions, measures for selecting the best split, Algorithm for decision tree induction.
Model over fitting: Due to presence of noise, due to lack of representation samples, evaluating the performance of classifier: hold out method, random subsampling, cross validation, bootstrap (Tan)

Classification: Basic concepts

Classification, **which** is **the task** of assigning objects **to** one of several **predefined** categories, **is a pervasive problem that encompasses many diverse applications. Examples include detecting spam email messages based upon the message header and content, categorizing cells as** malignant **or benign based upon the** results of MRI scans, and classifying galaxies based upon their shapes. A classification model may be built to categorize bank loan applications as either safe or risky, while a prediction model may be built to predict the expenditures of potential customers on computer equipment given their income and occupation.
Data classification is a two step process. In the first step, a model is built describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the class label attribute. In the second step the constructed model is used to classify the samples.
Classification is the task of learning a target function *f* that maps each attribute set X to one of the predefined class labels *y.*

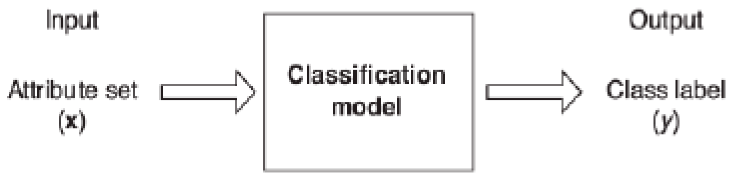


Figure Classification is the task of mapping an input attribute set x into its class label *y.*
Classification model is useful for the following purposes.
Descriptive Modeling A classification model can serve as an explanatory tool to distinguish between objects of different classes. For example, it would be useful-for both biologists and others-to have a descriptive model that summarizes the data shown in following Table and explains what features define a **vertebrate as a mammal, reptile, bird, fish, or amphibian.**

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

Table . The vertebrate data set.

Predictive Modeling A classification model can also be used to predict the class label of unknown records.

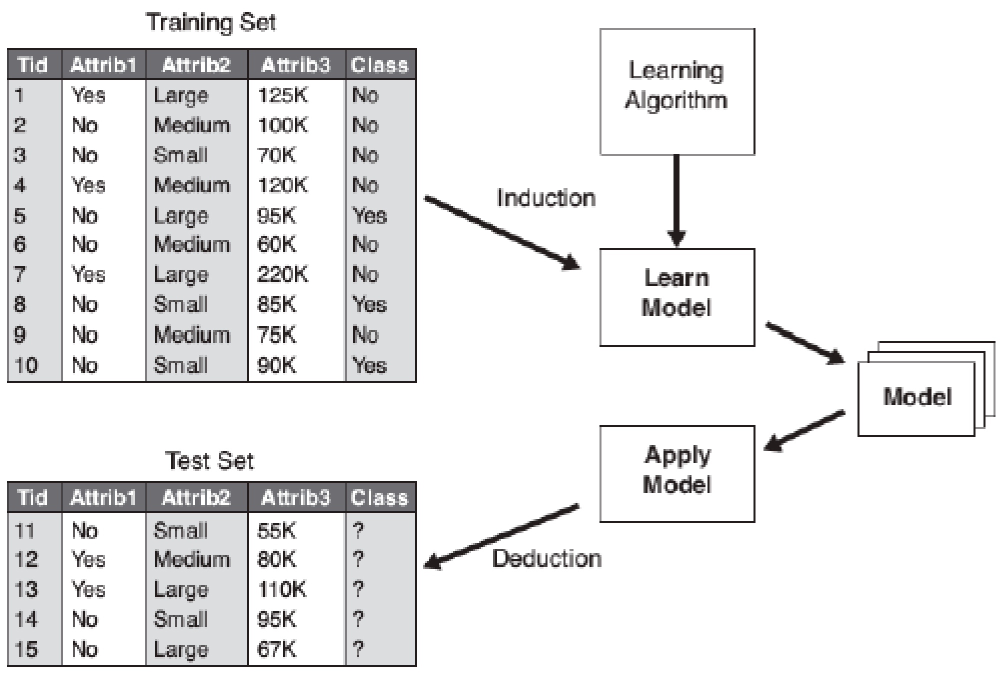
Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

General Approach to Solving a Classification Problem

A classification technique (or classifier) is a systematic approach to building classification models from an input data set. Examples include decision tree **classifiers, rule-based classifiers, neural networks, support vector machines,** and naive Bayes classifiers. Each technique employs a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previously unknown records.

The following figure shows a general approach for solving classification problems.

First, a training set consisting of records whose class labels are known must be provided. The training set is used to build a classification model, which is subsequently applied to the test set, which consists of records with unknown class labels.



Training Set

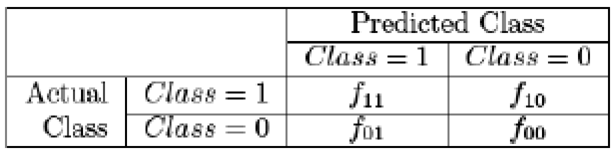
Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Test Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Figure: General approach for building a classification model.

Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a confusion matrix as shown below.



		Predicted Class	
		$Class = 1$	$Class = 0$
Actual Class	$Class = 1$	f_{11}	f_{10}
	$Class = 0$	f_{01}	f_{00}

Table: Confusion matrix for a 2 class problem.

Based on the entries in the confusion matrix, the total number of correct predictions made by the model is *(f11* + foo) and the total number of incorrect predictions is *(f1o + f01)* .

Although a confusion matrix provides the information needed to determine **how well a classification model performs, summarizing this information with a single number would make it** more **convenient to compare the performance** of different models. This can be done using a performance metric such as accuracy, which is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

Equivalently, the performance of a model can be expressed in terms of its error rate, which is given by the following equation:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set.

Decision Tree Induction

How a Decision Tree Works

To illustrate how classification with a decision tree works, consider a simpler version of the vertebrate classification problem described in the previous section.

Instead of classifying the vertebrates into five distinct groups of species, **we assign them to two categories: mammals and** non-mammals. Suppose a new species is discovered by scientists. How can we tell whether **it is a mammal or a non-mammal? One approach is to pose a series of questions** about the characteristics of the species. The first question we may ask is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. In the latter case, we need to ask a follow-up question: Do the females of the species give birth to their young? Those that do give birth are definitely mammals, while those that do not are likely to be non-mammals (with the exception of egg-laying mammals such as the platypus and spiny anteater).

The previous example illustrates how we can solve a classification problem by asking a series of carefully crafted questions about the attributes of the **test record. Each time we receive an answer, a follow-up question is asked** until we reach a conclusion about the class label of the record. The series of questions and their possible answers can be organized in the form of a decision tree, which is a hierarchical structure consisting of nodes and directed edges.

The following Figure shows the decision tree for the mammal classification problem. The tree has three types of nodes:

• A root node that has no incoming edges and zero or more outgoing edges.

• Internal nodes, each of which has exactly one incoming edge and two **or more outgoing edges.**

• Leaf or terminal nodes, each of which has exactly one incoming edge **and no outgoing edges.**

In a decision tree, each leaf node is assigned a class label. The nonterminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics.

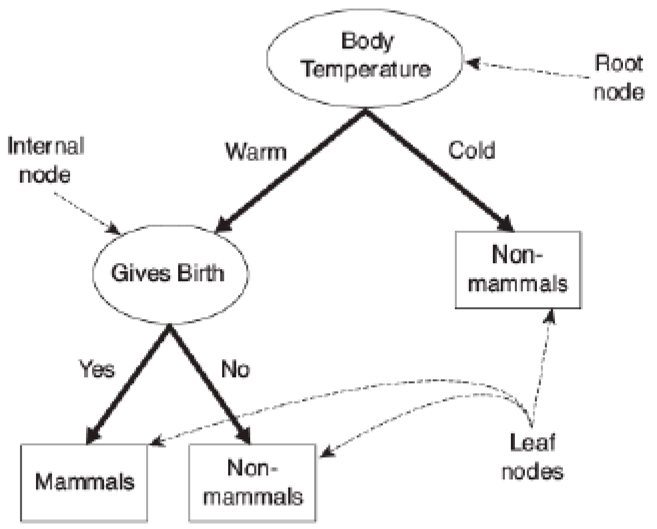
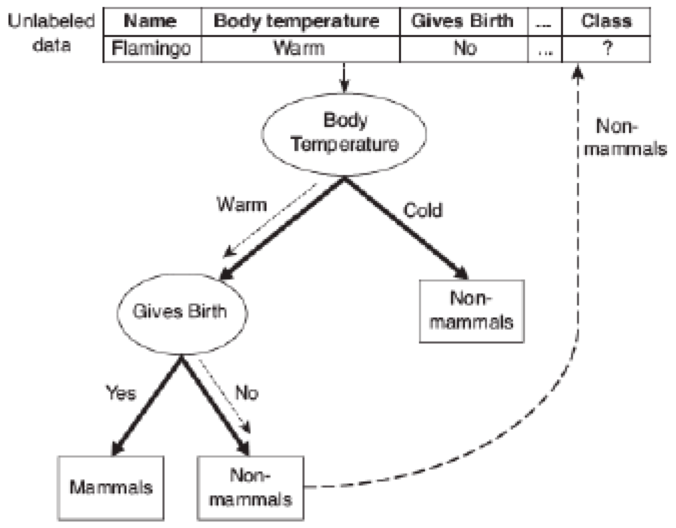


Figure: A decision tree for the mammal classification problem.

For example, the root node shown in above figure uses the attribute body Temperature to separate warm-blooded from cold-blooded vertebrates. Since all cold-blooded vertebrates are non-mammals, a leaf node labeled Non-mammals is created as the right child of the root node. If the vertebrate is warm-blooded, a subsequent attribute, Gives Birth, is used to distinguish mammals from other warm-blooded creatures, which are mostly birds.

Classifying a test record is straightforward once a decision tree has been constructed. Starting from the root node, we apply the test condition to the record and follow the appropriate branch based on the outcome of the test.



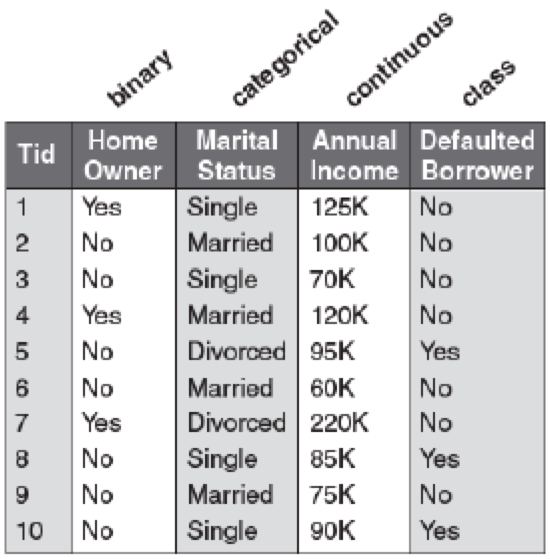
As an illustration, the above Figure traces the path in the decision tree that is used to predict the class label of a flamingo. The path terminates at a leaf node labeled Non-mammals.

<u>How to Build a Decision Tree</u>

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively purer subsets. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \ldots ,y_c\}$ be the class labels. The following is a recursive definition of Hunt's algorithm.

Step 1: If all the records in D_t, belong to the same class y_t then t is a leaf node labeled as y_t.
Step 2: If D_t contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.
In the example shown in following Figure each record contains the personal information of a borrower along with a class label indicating whether the borrower has defaulted **on loan payments.**


binary
categorical
continuous
class

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure: Training set for predicting borrowers who will default on loan payments.
The initial tree for the classification problem contains a single node with class label Defaulted = No *(see following* Figure (a)), which means that *most* of the borrowers successfully repaid their loans. The tree, however, needs to be refined since the root node contains records from both classes. The records M" subsequently divided into smaller subsets based on the outcomes of the Home Owner test condition, as shown in following Figure (b). Hunt's algorithm is then applied recursively to each child of the root node. From the training set given in above Figure, notice that all borrowers who are home owners successfully repaid their loans. The left child of the root is therefore a leaf node labeled Defaulted = No (see following Figure (b)). For the right child, we need to continue applying the recursive step of Hunt's algorithm until all

the records belong to the same class. The trees resulting from each recursive step are shown in following Figures (c) and (d).

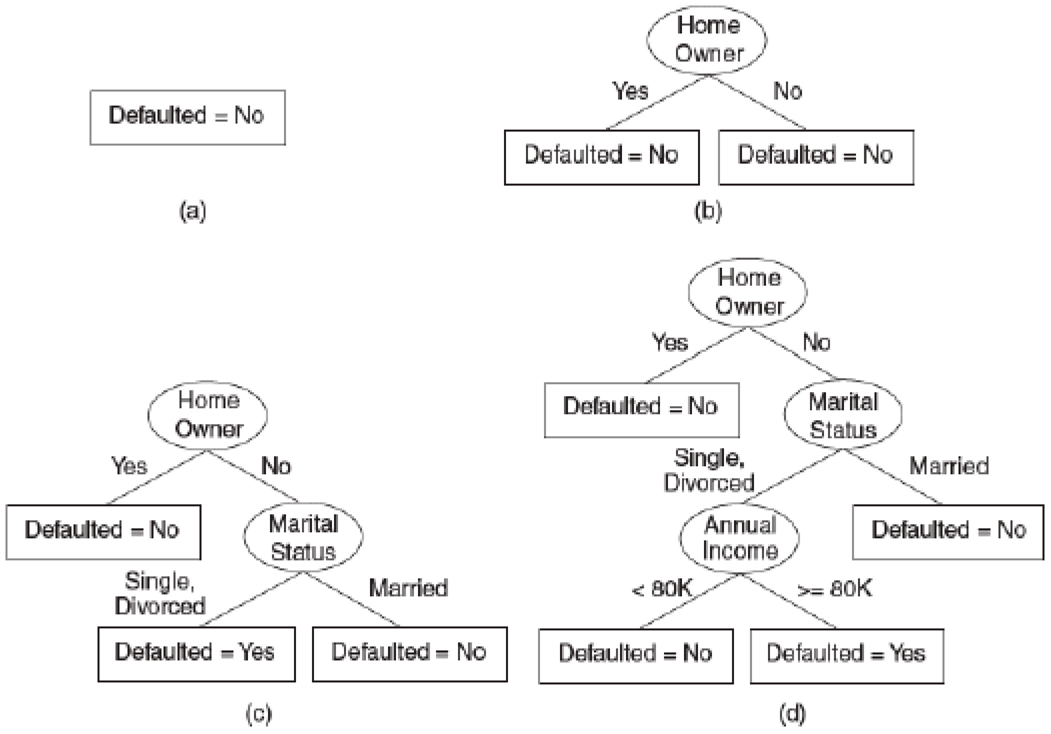


Figure: Hunts algorithm for inducing decision trees.

Design Issues of Decision Tree Induction

A learning algorithm for inducing decision trees must address the following two issues.

1. How should the training records be split? Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets.

2. How should the splitting procedure stop? A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values.

<u>Methods for Expressing Attribute Test Conditions</u>

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

<u>Binary Attributes</u> The test condition for a binary attribute generates two **potential outcomes, as shown in following Figure.**

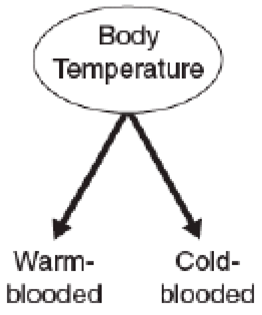
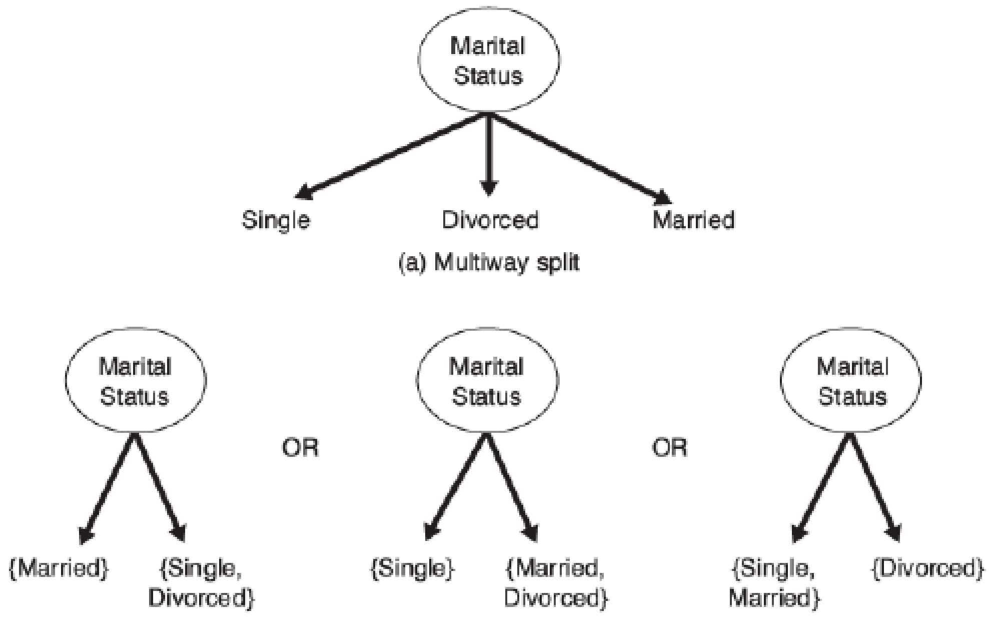


Figure: Test condition for binary attributes.

Nominal Attributes Since a nominal attribute can have many values its test condition can be expressed in two ways, as shown in following Figure. For a multi way split (following Figure (a), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values----single, married, or divorced-its test condition will produce a three-way split. On the other hand, some decision tree algorithms, such as CART, produce only binary splits by considering all 2^{k-1} - 1 ways of creating a binary partition of *k* attribute values. Following Figure (b) illustrates three different ways of grouping the attribute values for marital status into two subsets.



(b) Binary split {by grouping attribute values}

Figure: Test conditions for nominal attributes.

Ordinal Attributes Ordinal attributes can also produce binary or multi way splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. The following Figure illustrates various ways of splitting training records based on the Shirt Size attribute.

The groupings shown in Figures (a) and (b) preserve the order among the attribute values, whereas the grouping shown in Figure (c) violates this property because it combines the attribute values Small and Large into the same partition while Medium and Extra Large are combined into another **partition.**

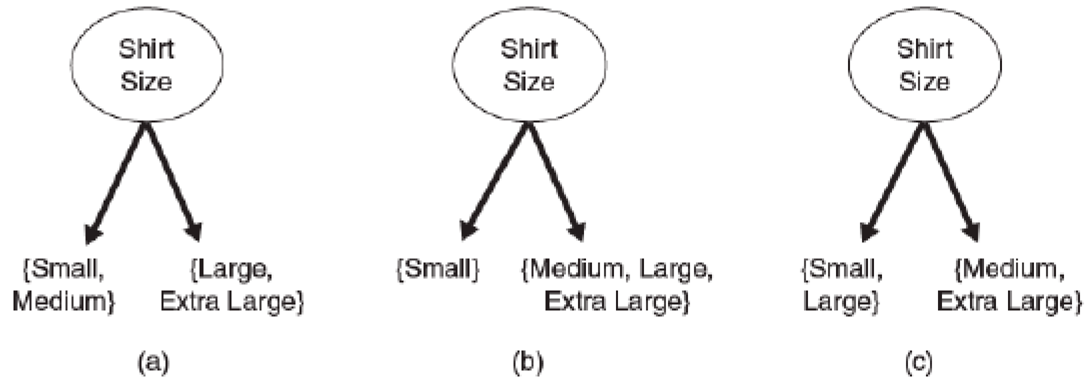


Figure: Different ways of grouping ordinal attribute values.

Continuous Attributes For continuous attributes, the test condition can be expressed as a comparison test *(A < v)* or *(A ≥ v)* with binary outcomes, or **a range query with outcomes of the form** *Vi ≤ A <* ***Vi+l,*** **for** i = **1, ... ,** ***k.*** **The** difference between these approaches is shown in following Figure

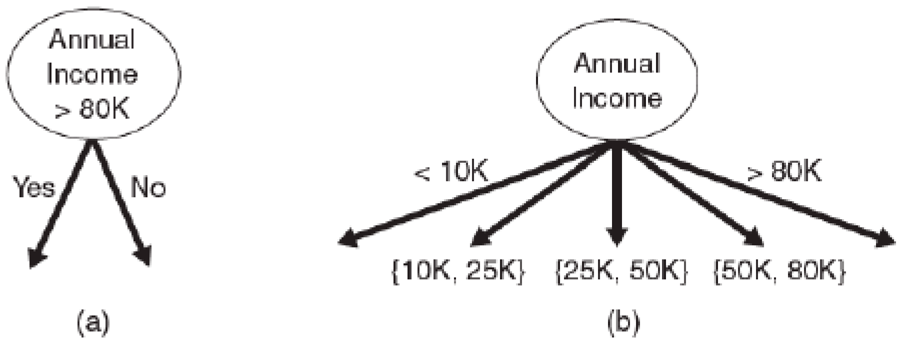


Figure: Test condition for continuous attributes.

Measures for Selecting the Best Split

Attribute selection measure:- The information gain measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an attribute selection measure or a measure of the goodness of split. The attribute with the highest information gain (or greatest entropy reduction) is chosen as the test attribute for the current node.

Let S be a set consisting of s data samples. Suppose the class label attribute has m distinct values defining m distinct classes, Ci (for i = 1,..,m). Let si be the number of samples of S in class Ci. The expected information needed to classify a given sample is given by:

$$I(s_1, s_2, \ldots, s_m) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

The entropy, or expected information based on the partitioning into subsets by A is given by:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \cdots + s_{mj}}{s} I(s_{1j}, \ldots, s_{mj}).$$

The smaller the entropy value is, the greater the purity of the subset partitions. The encoding information that would be gained by branching on A is

$$Gain(A) \quad = \quad I(s_1, s_2, \ldots, s_m) - E(A).$$

Example: Induction of a decision tree. The following Table presents a training set of data tuples taken from the All Electronics customer database. (The data are adapted from [Quinlan 1986b]). The class label attribute, buys computer, has two distinct values (namely {yes, no}), therefore, there are two distinct classes (m = 2). Let C1 correspond to the class yes and class C2 correspond to no. There are 9 samples of class yes and 5 samples of class no. To compute the information gain of each attribute, we first use first Equation to compute the expected information needed to classify a given sample. This is:

$$I(s_1, s_2) = I(9, 5) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

Next, we need to compute the entropy of each attribute. Let's start with the attribute age. We need to look at the distribution of yes and no samples for each value of age. We compute the expected information for each of these distributions.

rid	age	income	student	credit_rating	Class: buys_computer
1	<30	high	no	fair	no
2	<30	high	no	excellent	no
3	30-40	high	no	fair	yes
4	>40	medium	no	fair	yes
5	>40	low	yes	fair	yes
6	>40	low	yes	excellent	no
7	30-40	low	yes	excellent	yes
8	<30	medium	no	fair	no
9	<30	low	yes	fair	yes
10	>40	medium	yes	fair	yes
11	<30	medium	yes	excellent	yes
12	30-40	medium	no	excellent	yes
13	30-40	high	yes	fair	yes
14	>40	medium	no	excellent	no

Table. Training set

Using second equation the expected information needed to classify a given sample if the samples are partitioned according to age is

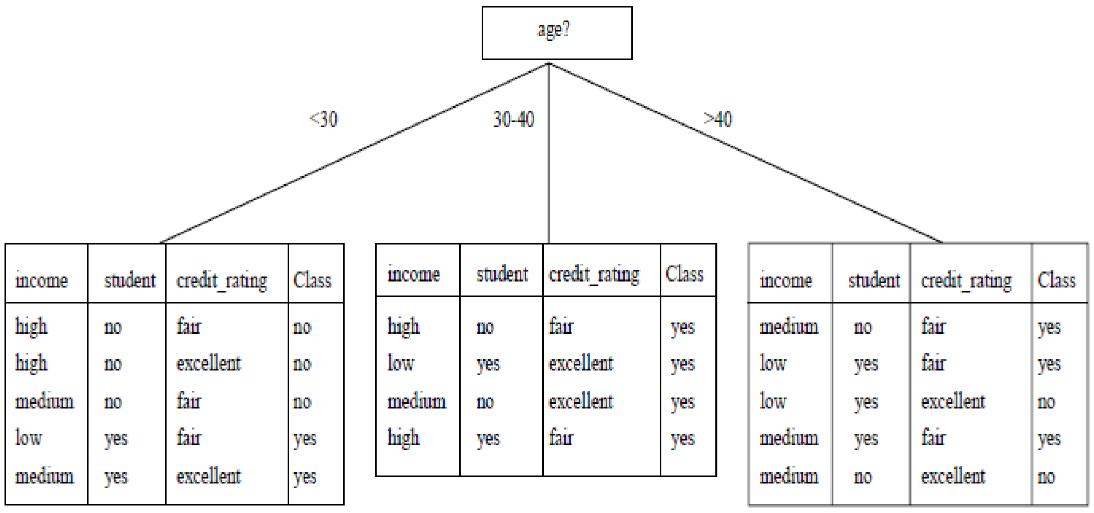
for age = "<30":	$s_{11} = 2$	$s_{21} = 3$	$I(s_{11}, s_{21}) = 0.971$
for age = "30-40":	$s_{12} = 4$	$s_{22} = 0$	$I(s_{12}, s_{22}) = 0$
for age = ">40":	$s_{13} = 3$	$s_{23} = 2$	$I(s_{13}, s_{23}) = 0.971$

$$E(age) = \frac{5}{14}I(s_{11}, s_{21}) + \frac{4}{14}I(s_{12}, s_{22}) + \frac{5}{14}I(s_{13}, s_{23}) = 0.694.$$

Hence, the gain in information from such a partition would be:

$$Gain(age) = I(s_1, s_2) - E(age) = 0.246$$

Similarly, we can compute Gain (income) = 0.029, Gain(student) = 0.151, and Gain(credit rating) = 0.048. Since age has the highest information gain among the attributes, it is selected as the test attribute. A node is created and labeled with age, and branches are grown for each of the attribute's values. The samples are then partitioned accordingly, as shown in following Figure.

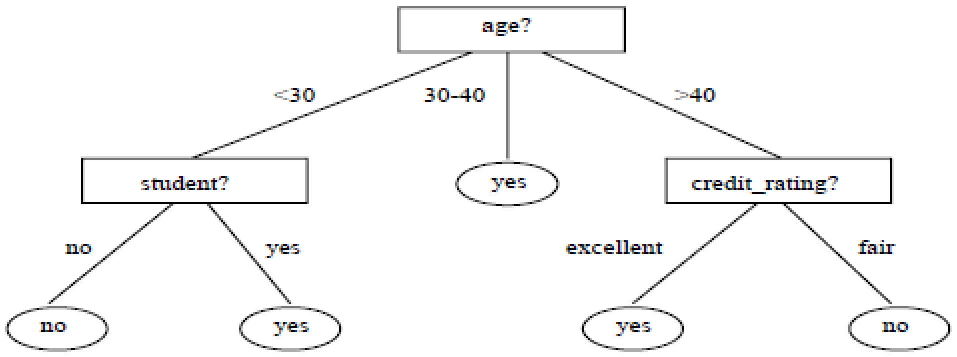


income	student	credit_rating	Class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	Class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	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

Notice that the samples falling into the partition for age = 30-40 all belong to the same class. Since they all belong to class yes, a leaf should therefore be created at the end of this branch and labeled with yes. The final decision tree returned by the algorithm is shown in following Figure.



Algorithm: (generate_decision_tree) generate a decision tree from the given training data

Input: The training samples, *samples*, represented by discrete-valued attributes; the set of candidate attributes, *attribute-list*.

Output: A decision tree.

Method:

```
1) create a node N;
2) if samples 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 most common class in samples; // majority voting
6) select test-attribute, the attribute among attribute-list with the highest information gain;
7) label node N with test-attribute;
8) for each known value a_i of test-attribute // partition the samples
9)     grow a branch from node N for the condition test-attribute=a_i;
10)    let s_i be the set of samples in samples for which test-attribute=a_i; // a partition
11)    if s_i is empty then
12)            attach a leaf labeled with the most common class in samples;
13)    else attach the node returned by Generate_decision_tree(s_i, attribute-list - test-attribute);
```

□

Some examples of impurity measures include

$$\begin{aligned}\text{Entropy}(t) &= -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t),\\ \text{Gini}(t) &= 1-\sum_{i=0}^{c-1}[p(i|t)]^2,\\ \text{Classification error}(t) &= 1-\max_i[p(i|t)],\end{aligned}$$

where c is the number of classes and 0 log2 0 = 0 in entropy calculations
We provide several examples of computing the
different impurity measures.

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1-(0/6)^2-(6/6)^2 = 0$$
$$\text{Entropy} = -(0/6)\log_2(0/6)-(6/6)\log_2(6/6) = 0$$
$$\text{Error} = 1-\max[0/6, 6/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Gini} = 1-(1/6)^2-(5/6)^2 = 0.278$$
$$\text{Entropy} = -(1/6)\log_2(1/6)-(5/6)\log_2(5/6) = 0.650$$
$$\text{Error} = 1-\max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1-(3/6)^2-(3/6)^2 = 0.5$$
$$\text{Entropy} = -(3/6)\log_2(3/6)-(3/6)\log_2(3/6) = 1$$
$$\text{Error} = 1-\max[3/6, 3/6] = 0.5$$

Splitting of Binary Attributes

Consider the diagram shown in following Figure. Suppose there are two ways to split the data into smaller subsets. Before splitting, the Gini index is 0.5 since there are an equal number of records from both classes. If attribute *A* is chosen to split the data, the Gini index for node N1 is 0.4898, and for node N2, it is 0.480. The weighted average of the Gini index for the descendent nodes is (7/12) x 0.4898 + (5/12) x 0.480 = 0.486. Similarly, we can show that the weighted average of the Gini index for attribute *B* is 0.375. Since the subsets for attribute *B* have a smaller Gini index, it is preferred over attribute *A.*

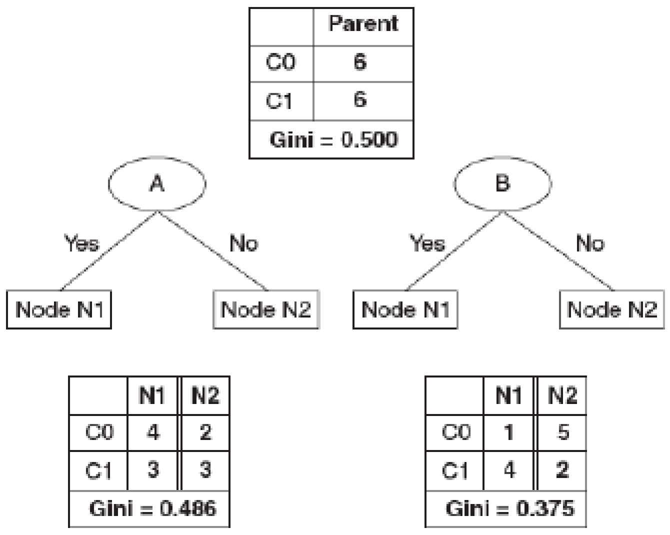


Figure: Splitting binary attributes.

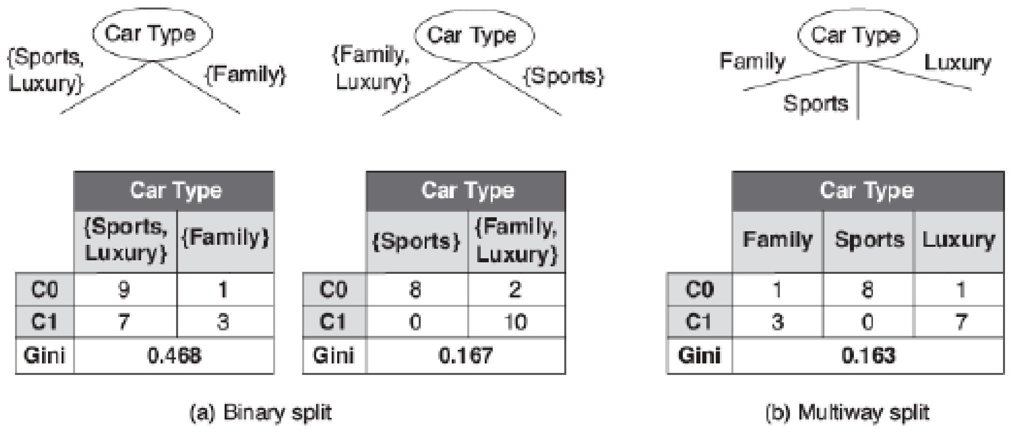
<u>Splitting of Nominal Attributes</u>

As previously noted, a nominal attribute can produce either binary or multi way splits. The computation of the Gini index for a binary split is similar to that shown for determining binary attributes. For the first binary grouping of the Car Type attribute, the Gini index of {Sports, Luxury} is 0.4922 and the Gini index of {Family} is 0.3750. The weighted average Gini index for the grouping is equal to 16/20 X 0.4922 +4/20 X 0.3750 = 0.468.

Similarly, for the second binary grouping of {Sports} and {Family. Luxury}, the weighted average Gini index is 0.167. The second grouping has a lower Gini index because its corresponding subsets are much purer.

For the multiway split, the Gini index is computed for every attribute value. Since Gini({Family}) = 0.375, Gini({Sports}) = 0, and Gini({Luxury}) = 0.219, the overall Gini index for the multi way split is equal to 4/20 X 0.375 + 8/20 X 0 + 8/20 X 0.219 = 0.163.

The multiway split has a smaller Gini index compared to both two-way splits.



	Car Type	
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	

	Car Type	
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

(a) Binary split

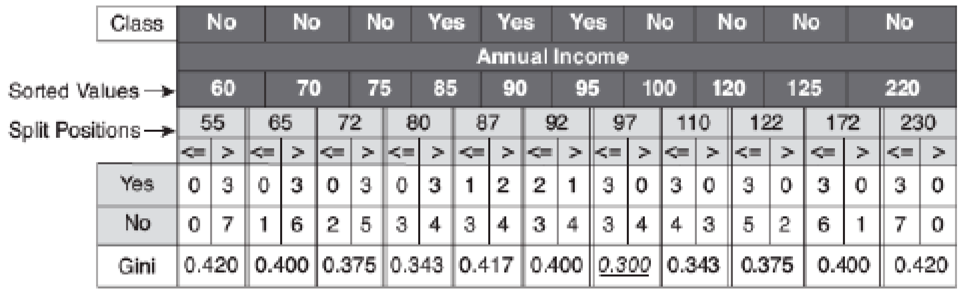
	Car Type		
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(b) Multiway split

Figure Splitting nominal attributes.

<u>Splitting of Continuous Attributes</u>

Consider the example shown in Figure 4.16, in which the test condition Annual Income ≤ *v* is used to split the training records for the loan default classification problem.



Class		No		No		No		Yes		Yes		Yes		No		No		No		No		
	Annual Income																					
Sorted Values		60		70		75		85		90		95		100		120		125		220		
Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		<u>0.300</u>		0.343		0.375		0.400		0.420	

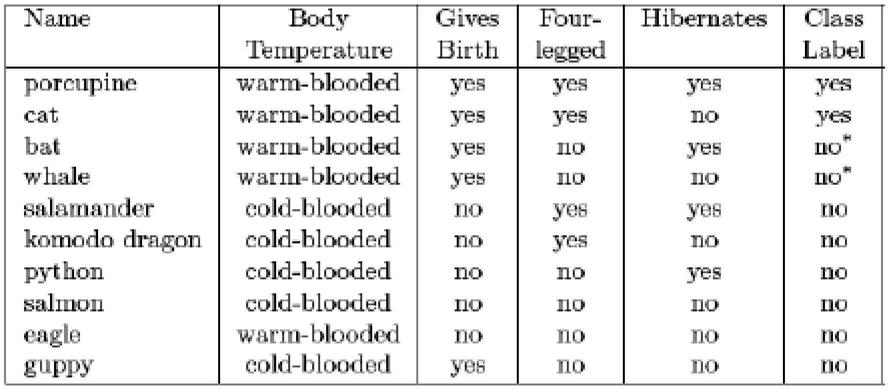
Figure: Splitting continuous attributes.

The best split position corresponds to the one that produces the smallest Gini index, i.e., *v* = 97. This procedure *is* less **expensive because it requires a constant amount of time to update the class** distribution at each candidate split position.

<u>Model Over fitting</u>

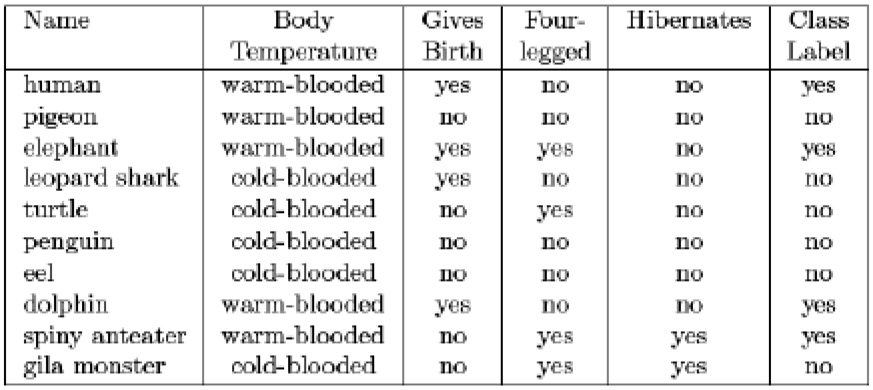
The errors committed by a classification model are generally divided into two types: training errors and generalization errors. Training error, also known as (resubstitution error or apparent error) is the number of misclassification errors committed on training records) whereas generalization error is the expected error of the model on previously unseen records. A good model must have low training error *as* well as low generalization error. This is important because a model that fits the training data too well can have a poorer generalization error than a model with a higher training error. Such a situation is known as model over fitting.

Notice that the training and test error rates of the model are large when the size of the tree is very small. This situation is known as model under fitting. As the number of nodes in the decision tree increases, the tree will have **fewer training and test errors. However, once the tree becomes too large, its test error rate begins to** increase **even though its training error rate continues** to decrease. This phenomenon is known as model over fitting. Over fitting Due to Presence of Noise Consider the training and test sets shown in following Tables for the mammal classification problem. Two of the ten training records are mislabeled: bats **and whales are classified as non-mammals instead of** mammals.



Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Table: An example of training set for classifying mammals. Class labels with asterisk symbols represent mislabeled records.



Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	cold-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

Table: An example test set for classifying mammals.

A decision tree that perfectly fits the training data is shown in Figure (a). Although the training error for the tree is zero, its error rate on the test set is 30%. Both humans and dolphins were misclassified as non-mammals because their attribute values for Body Temperature, Gives Birth, and Four-legged are identical to the mislabeled records in the training set. In contrast, the decision tree *M2* shown in Figure 4.25(b) has a lower test error rate (10%) even though its training error rate is somewhat higher (20%).

It is evident that the first decision tree, MI, has over fitted the training data because there is a simpler model with lower error rate on the test set. The Four-legged attribute test condition in model Ml is spurious because it fits the mislabeled training records, which leads to the misclassification of records in the test set.

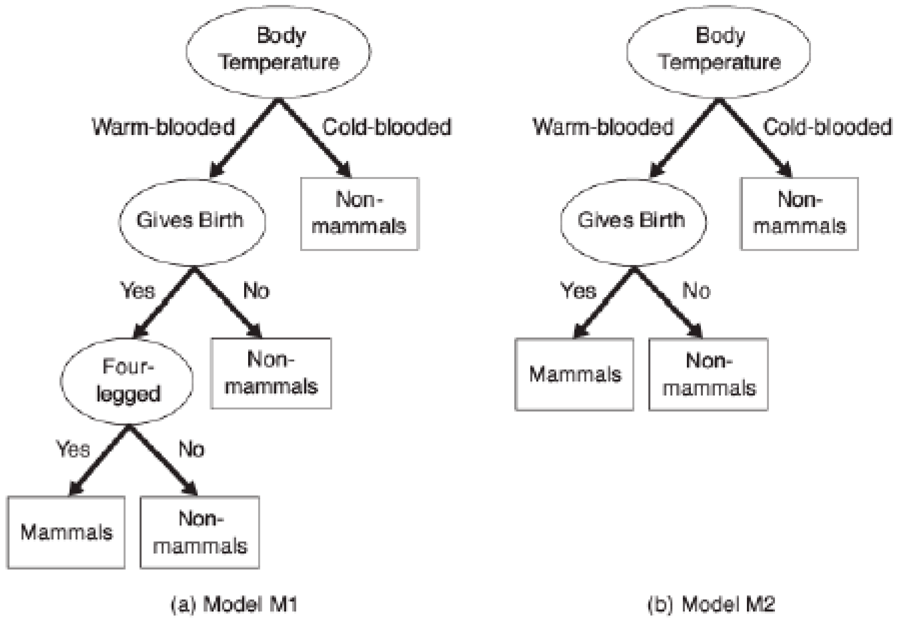


Figure: Decision tree induced from the data set shown in above Table

Over fitting Due to Lack of Representative Samples

Models that make their classification decisions based on a small number of training records are also susceptible to overfitting. Such models can be gener**ated because of lack of representative samples in the training data and learning algorithms that continue to refine their models even when few training records** are available.

Consider the five training records shown in following Table. All of these training records are labeled correctly and the corresponding decision tree is depicted in following Figure. Although its training error is zero, its error rate on the test set is 30%.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
salamander	cold-blooded	no	yes	yes	no
guppy	cold-blooded	yes	no	no	no
eagle	warm-blooded	no	no	no	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes

Table: An example training set for classifying mammals.

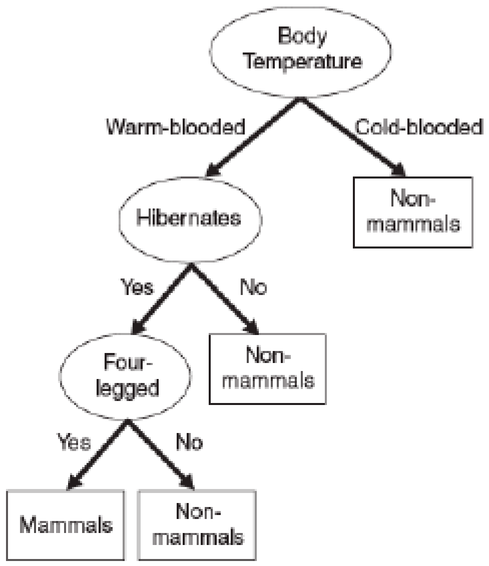


Figure: Decision tree induced from the data set shown in above table

Humans, elephants, and dolphins are misclassified because the decision tree classifies all warm-blooded vertebrates that do not hibernate as non-mammals.

The tree arrives at this classification decision because there is only one training record, which is an eagle, with such characteristics. This example clearly demonstrates the danger of making wrong predictions when there are not enough representative examples at the leaf nodes of a decision tree.

<u>Evaluating the Performance of a Classifier</u>

<u>Holdout Method</u>

In the holdout method, the original data with labeled examples is partitioned into two disjoint sets, called the training and the test sets, respectively. A classification model is then induced from the training set and its performance is evaluated on the test set. The proportion of data reserved for training and for testing is typically at the discretion of the analysts (e.g., 50-50 or two thirds for training and one-third for testing). The accuracy of the classifier can be estimated based on the accuracy of the induced model on the test set.

The holdout method has several well-known limitations.

First, fewer labeled examples are available for training because some of the records are withheld for testing. As a result, the induced model may not be as good as when all the labeled examples are used for training.

Second, the model may be highly dependent on the composition of the training and test sets. Smaller the training set size, the larger the variance of the model. On the other hand, if the training set is too large, then the estimated accuracy computed from the smaller test set is less reliable.

Finally, the training and test sets are no longer independent of each other. Because the training and test sets are subsets of the original data, a class that is overrepresented in one subset will be underrepresented in the other, and vice versa.

Random Subsampling

The holdout method can be repeated several times to improve the estimation of a classifier's performance. This approach is known as random subsampling. Let acc_i be the model accuracy during the *ith* iteration. The overall accuracy is given by $Acc_{sub} = \sum_{i=1}^{k} acc_i / k$. Random subsampling still encounters some of the problems associated with the holdout method because it does not utilize as much data as possible for training. It also has no control over the number of times each record is used for testing and training. Consequently, some records might be used for training more often than others.

Cross-Validation

An alternative to random subsampling is cross-validation. In this approach, each record is used the same number of times for training and exactly once for testing. To illustrate this method, suppose we partition the data into two equal-sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called a twofold cross-validation. The total error is obtained by summing up the errors for both runs. The k-fold cross-validation method generalizes this approach by segmenting the data into *k* equal-sized partitions. During each run, one of the partitions is chosen for testing, while the rest of them are used for training.

This procedure is repeated *k* times so that each partition is used for testing exactly once. Again, the total error is found by summing up the errors for all *k* runs. The drawback of this approach is that it is computationally expensive to repeat the procedure *N* **times.**

Bootstrapapproach

The methods presented so far assume that the training records are sampled without replacement. *As* a result, there are no duplicate records in the training and test sets. In the bootstrap approach, the training records are sampled with replacement; i.e., a record already chosen for training is put back into the original pool of records so that it is equally likely to be redrawn.