Lecture 6 - Decision Trees

Adnan Ferdous Ashrafi

Stamford University Bangladesh

Table of Contents

Definition of Decision Trees

Definition

One attractive classification method involves the construction of a decision tree, a collection of *decision nodes*, connected by *branches*, extending downward from the *root node* until terminating in *leaf nodes*. Beginning at the root node, which by convention is placed at the top of the decision tree diagram, *attributes* are tested at the decision nodes, with each possible outcome resulting in a *branch*. Each branch then leads either to another *decision node* or to a terminating *leaf node*.

Figure 1 provides an example of a simple decision tree.

Definition of Decision Trees

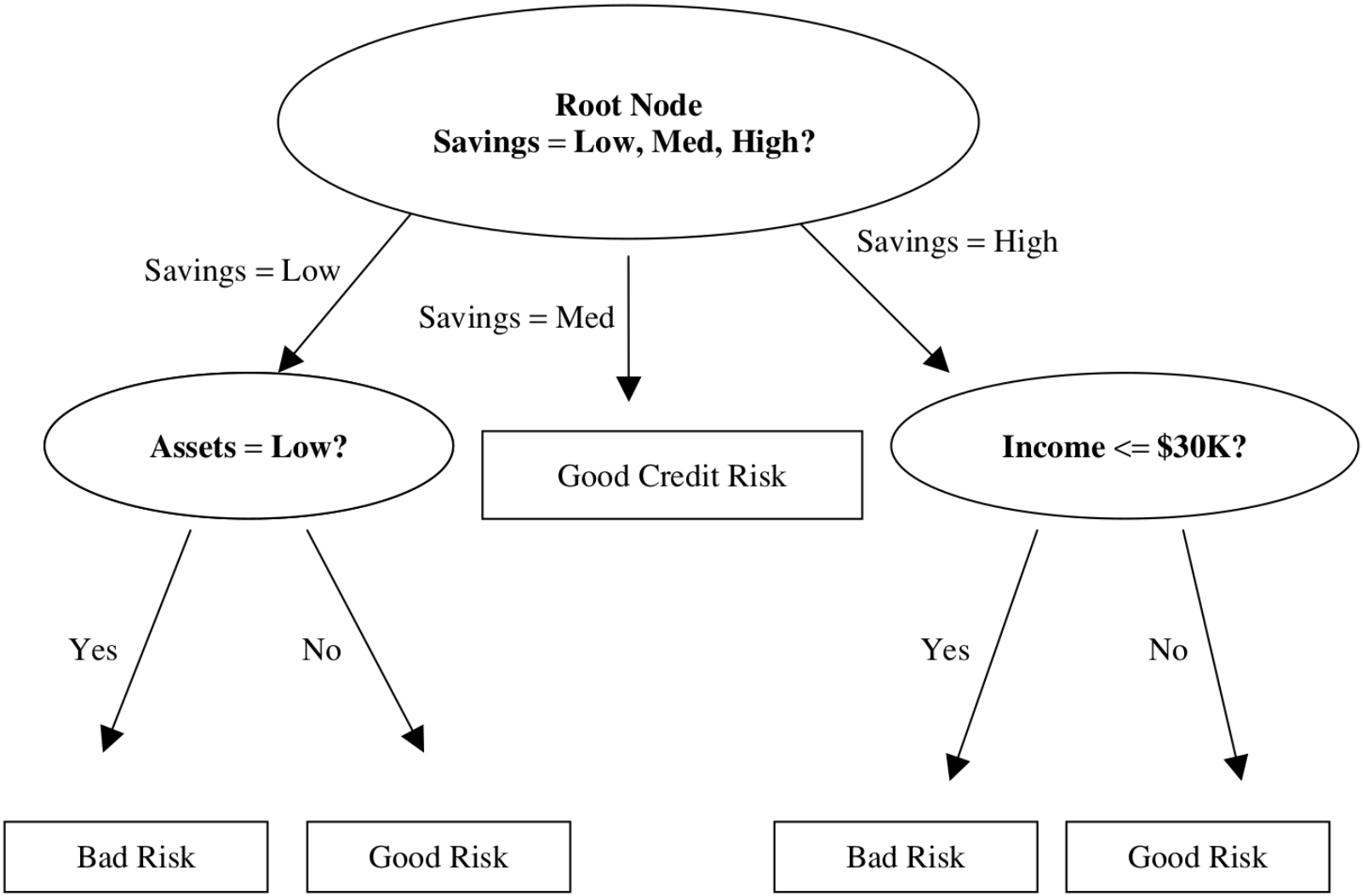

Figure 1: Simple decision tree

Example of Decision Trees

The target variable for the decision tree in Figure 1 is *credit risk*, with potential customers being classified as either good or bad credit risks. The predictor variables are *savings* (low, medium, and high), *assets* (low or not low), and *income* ($\leq \$50,000$ or $> \$50,000$). Here, the root node represents a **decision node**, testing whether each record has a low, medium, or high savings level (as defined by the analyst or domain expert). The data set is partitioned, or *split*, according to the values of this attribute. Those records with low savings are sent via the leftmost branch (*savings* = *low*) to another decision node. The records with high savings are sent via the rightmost branch to a different decision node.

Example of Decision Trees

The records with medium savings are sent via the middle branch directly to a leaf node, indicating the termination of this branch.

Why a leaf node and not another decision node?

Because, in the data set (not shown), all of the records with medium savings levels have been classified as good credit risks. There is no need for another decision node, because our knowledge that the customer has medium savings predicts good credit with 100% accuracy in the data set.

Example of Decision Trees

For customers with low savings, the next decision node tests whether the customer has low assets. Those with low assets are then classified as bad credit risks; the others are classified as good credit risks.

For customers with high savings, the next decision node tests whether the customer has an income of at most $30,000. Customers with incomes of $30,000 or less are then classified as bad credit risks, with the others classified as good credit risks.

Example of Decision Trees

When no further splits can be made, the decision tree algorithm stops growing new nodes. For example, suppose that all of the branches terminate in "**pure**" leaf nodes, where the target variable is unary for the records in that node (e.g., each record in the leaf node is a good credit risk). Then no further splits are necessary, so no further nodes are grown.

Example of Decision Trees

However, there are instances when a particular node contains "**diverse**" attributes (with non-unary values for the target attribute), and yet the decision tree cannot make a split.

For example, suppose that we consider the records from Figure 1 with high savings and low income ($\leq \$30,000$). Suppose that there are five records with these values, all of which also have low assets. Finally, suppose that three of these five customers have been classified as bad credit risks and two as good credit risks, as shown in Table 1. In the real world, one often encounters situations such as this, with varied values for the response variable, even for exactly the same values for the predictor variables.

Example of Decision Trees

Here, since all customers have the same predictor values, there is no possible way to split the records according to the predictor variables that will lead to a pure leaf node. Therefore, such nodes become diverse leaf nodes, with mixed values for the target attribute. In this case, the decision tree may report that the classification for such customers is "**bad**," with 60% confidence, as determined by the three-fifths of customers in this node who are bad credit risks. Note that not all attributes are tested for all records. Customers with low savings and low assets, for example, are not tested with regard to income in this example.

Example of Decision Trees

Table 1: Sample of Records That Cannot Lead to Pure Leaf Node

Customer	Savings	Assets	Income	Credit Risk
004	High	Low	$\leq$ \$30000	Good
009	High	Low	$\leq$ \$30000	Good
027	High	Low	$\leq$ \$30000	Bad
031	High	Low	$\leq$ \$30000	Bad
104	High	Low	$\leq$ \$30000	Bad

Requirements before applying Decision Trees

Certain requirements must be met before decision tree algorithms may be applied:

1. Decision tree algorithms represent supervised learning, and as such require preclassified target variables. A training data set must be supplied which provides the algorithm with the values of the target variable.
2. This training data set should be rich and varied, providing the algorithm with a healthy cross section of the types of records for which classification may be needed in the future. Decision trees learn by example, and if examples are systematically lacking for a definable subset of records, classification and prediction for this subset will be problematic or impossible.
3. The target attribute classes must be discrete. That is, one cannot apply decision tree analysis to a continuous target variable. Rather, the target variable must take on values that are clearly demarcated as either belonging to a particular class or not belonging.

Why does Decision Trees behave like this?

Why in the example above, did the decision tree choose the savings attribute for the root node split?

Why did it not choose assets or income instead?

Decision trees seek to create a set of leaf nodes that are as "pure" as possible, that is, where each of the records in a particular leaf node has the same classification. In this way, the decision tree may provide classification assignments with the highest measure of confidence available.

However, how does one measure uniformity, or conversely, how does one measure heterogeneity?

Types of Decision Trees

We shall examine two of the many methods for measuring leaf node purity, which lead to the two leading algorithms for constructing decision trees:

- Classification and regression trees (CART) algorithm
- C4.5 algorithm

CLASSIFICATION AND REGRESSION TREES

The *classification and regression trees* (CART) method was suggested by Breiman et al. [1] in 1984. The decision trees produced by CART are strictly binary, containing exactly two branches for each decision node. CART recursively partitions the records in the training data set into subsets of records with similar values for the target attribute. The CART algorithm grows the tree by conducting for each decision node, an exhaustive search of all available variables and all possible splitting values, selecting the optimal split according to the following criteria (from Kennedy et al. [2]).

CLASSIFICATION AND REGRESSION TREES

Let $\Phi(s|t)$ be a measure of the "goodness" of a candidate split s at node t, where,

$$\Phi(s|t) = 2P_L P_R \sum_{j=1}^{\#\text{classes}} |P(j|t_L) - P(j|t_R)| \tag{1}$$

and where,

$$\begin{aligned}
t_L &= \text{left child node of node } t \\
t_R &= \text{right child node of node } t \\
P_L &= \frac{\text{number of records at } t_L}{\text{number of records in training set}} \\
P_R &= \frac{\text{number of records at } t_R}{\text{number of records in training set}} \\
P(j|t_L) &= \frac{\text{number of class } j \text{ records at } t_L}{\text{number of records at } t} \\
P(j|t_R) &= \frac{\text{number of class } j \text{ records at } t_R}{\text{number of records at } t}
\end{aligned}$$

Then the optimal split is whichever split maximizes this measure $\Phi(s|t)$ over all possible splits at node t.

CART - Example

Suppose that we have the training data set shown in Table 2 and are interested in using CART to build a decision tree for predicting whether a particular customer should be classified as being a good or a bad credit risk. In this small example, all eight training records enter into the root node. Since CART is restricted to binary splits, the candidate splits that the CART algorithm would evaluate for the initial partition at the root node are shown in Table 3. Although income is a continuous variable, CART may still identify a finite list of possible splits based on the number of different values that the variable actually takes in the data set. Alternatively, the analyst may choose to categorize the continuous variable into a smaller number of classes.

CART - Example - Training set

Customer	Savings	Assets	Income ($1000s)	Credit Risk
1	Medium	High	75	Good
2	Low	Low	50	Bad
3	High	Medium	25	Bad
4	Medium	Medium	50	Good
5	Low	Medium	100	Good
6	High	High	25	Good
7	Low	Low	25	Bad
8	Medium	Medium	75	Good

Figure 2: Training Set of Records for Classifying Credit Risk

Candidate Split	Left Child Node, t_L	Right Child Node, t_R
1	$Savings = low$	$Savings \in \{medium, high\}$
2	$Savings = medium$	$Savings \in \{low, high\}$
3	$Savings = high$	$Savings \in \{low, medium\}$
4	$Assets = low$	$Assets \in \{medium, high\}$
5	$Assets = medium$	$Assets \in \{low, high\}$
6	$Assets = high$	$Assets \in \{low, medium\}$
7	$Income \leq \$25{,}000$	$Income > \$25{,}000$
8	$Income \leq \$50{,}000$	$Income > \$50{,}000$
9	$Income \leq \$75{,}000$	$Income > \$75{,}000$

Figure 3: Candidate Splits for t = Root Node

CART - Example - Candidate Splits

For each candidate split, let us examine the values of the various components of the optimality measure $\Phi(s|t)$ in Table 6.4. Using these observed values, we may investigate the behavior of the optimality measure under various conditions.

For example, when is $\Phi(s|t)$ large?

We see that $\Phi(s|t)$ is large when both of its main components are large: $2P_L P_R$ and $\sum_{j=1}^{\#\text{classes}} |P(j|t_L) - P(j|t_R)|$.

CART - Example - Candidate Splits

Split	P_L	P_R	$P(j\|t_L)$	$P(j\|t_R)$	$2P_LP_R$	$Q(s\|t)$	$\Phi(s\|t)$
1	0.375	0.625	G: .333 B: .667	G: .8 B: .2	0.46875	0.934	0.4378
2	0.375	0.625	G: 1 B: 0	G: 0.4 B: 0.6	0.46875	1.2	0.5625
3	0.25	0.75	G: 0.5 B: 0.5	G: 0.667 B: 0.333	0.375	0.334	0.1253
4	0.25	0.75	G: 0 B: 1	G: 0.833 B: 0.167	0.375	1.667	0.6248
5	0.5	0.5	G: 0.75 B: 0.25	G: 0.5 B: 0.5	0.5	0.5	0.25
6	0.25	0.75	G: 1 B: 0	G: 0.5 B: 0.5	0.375	1	0.375
7	0.375	0.625	G: 0.333 B: 0.667	G: 0.8 B: 0.2	0.46875	0.934	0.4378
8	0.625	0.375	G: 0.4 B: 0.6	G: 1 B: 0	0.46875	1.2	0.5625
9	0.875	0.125	G: 0.571 B: 0.429	G: 1 B: 0	0.21875	0.858	0.1877

Figure 4: Values of the Components of the Optimality Measure $\Phi(s|t)$ for Each Candidate Split, for the Root Node

CART - Example - Candidate Splits

Let $Q(s|t) = \sum_{j=1}^{\#\text{classes}} |P(j|t_L) - P(j|t_R)|$.

When is the component $Q(s|t)$ large?

$Q(s|t)$ is large when the distance between $P(j|t_L)$ and $P(j|t_R)$ is maximized across each class (value of the target variable). In other words, this component is maximized when the proportions of records in the child nodes for each particular value of the target variable are as different as possible. The maximum value would therefore occur when for each class the child nodes are completely uniform (pure). The theoretical maximum value for $Q(s|t)$ is k, where k is the number of classes for the target variable. Since our output variable credit risk takes two values, good and bad, $k = 2$ is the maximum for this component.

CART - Example - Candidate Splits

The component $2P_L P_R$ is maximized when P_L and P_R are large, which occurs when the proportions of records in the left and right child nodes are equal.

Therefore, $\Phi(s|t)$ will tend to favor balanced splits that partition the data into child nodes containing roughly equal numbers of records. Hence, the optimality measure $\Phi(s|t)$ prefers splits that will provide child nodes that are homogeneous for all classes and have roughly equal numbers of records.

The theoretical maximum for $2P_L P_R$ is $2(0.5)(0.5) = 0.5$.

CART - Example - Candidate Splits

In this example, only candidate split 5 has an observed value for $2P_L P_R$ that reaches the theoretical maximum for this statistic, 0.5, since the records are partitioned equally into two groups of four. The theoretical maximum for $Q(s|t)$ is obtained only when each resulting child node is pure, and thus is not achieved for this data set.

CART - Example - Initial Split

The maximum observed value for $\Phi(s|t)$ among the candidate splits is therefore attained by split 4, with $\Phi(s|t) = 0.6248$. CART therefore chooses to make the initial partition of the data set using candidate split 4, assets = low versus assets $\in$ {medium, high}, as shown in Figure 5.

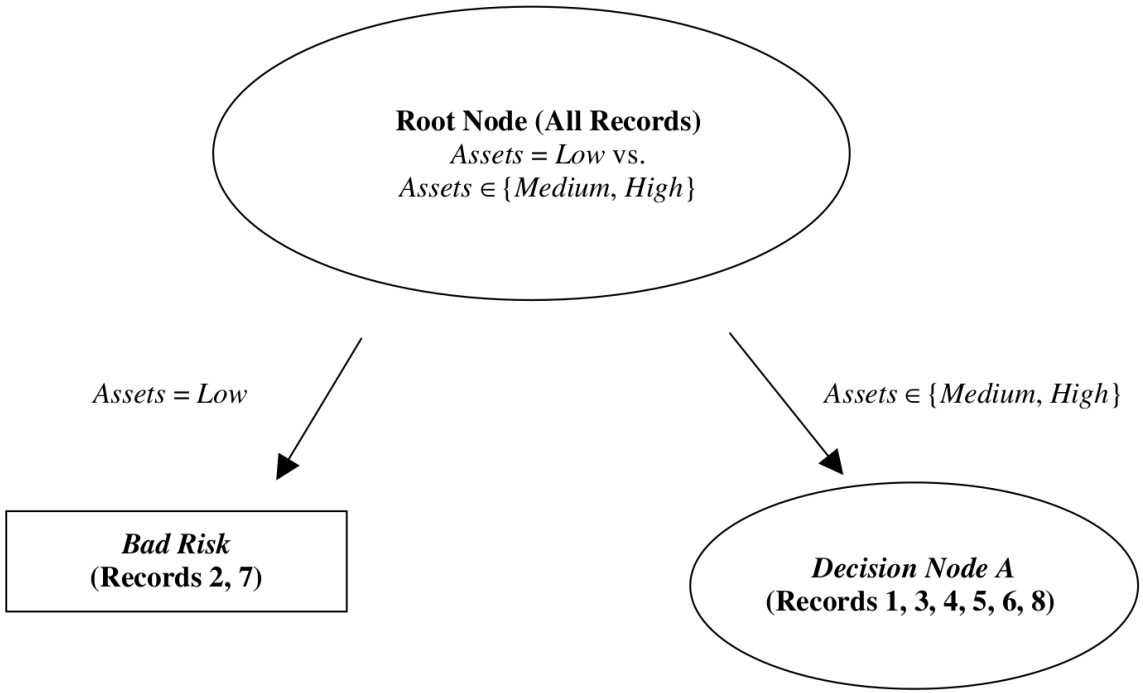

Figure 5: CART decision tree after initial split

CART - Example - After the Initial Split

The left child node turns out to be a terminal leaf node, since both of the records that were passed to this node had bad credit risk. The right child node, however, is diverse and calls for further partitioning. We again compile a table of the candidate splits (all are available except split 4), along with the values for the optimality measure (Figure 6). Here two candidate splits (3 and 7) share the highest value for $\Phi(s|t)$, 0.4444. We arbitrarily select the first split encountered, split 3, savings = high versus savings $\in$ {low, medium}, for decision node A, with the resulting tree shown in Figure 7.

CART - Example - Second Split

Split	P_L	P_R	$P(j\|t_L)$	$P(j\|t_R)$	$2P_L P_R$	$Q(s\|t)$	$\Phi(s\|t)$
1	0.167	0.833	G: 1 B: 0	G: .8 B: .2	0.2782	0.4	0.1112
2	0.5	0.5	G: 1 B: 0	G: 0.667 B: 0.333	0.5	0.6666	0.3333
3	0.333	0.667	G: 0.5 B: 0.5	G: 1 B: 0	0.4444	1	0.4444
5	0.667	0.333	G: 0.75 B: 0.25	G: 1 B: 0	0.4444	0.5	0.2222
6	0.333	0.667	G: 1 B: 0	G: 0.75 B: 0.25	0.4444	0.5	0.2222
7	0.333	0.667	G: 0.5 B: 0.5	G: 1 B: 0	0.4444	1	0.4444
8	0.5	0.5	G: 0.667 B: 0.333	G: 1 B: 0	0.5	0.6666	0.3333
9	0.167	0.833	G: 0.8 B: 0.2	G: 1 B: 0	0.2782	0.4	0.1112

Figure 6: Values of the Components of the Optimality Measure $\Phi(s|t)$ for Each Candidate Split, for Decision Node A

CART - Example - Second Split

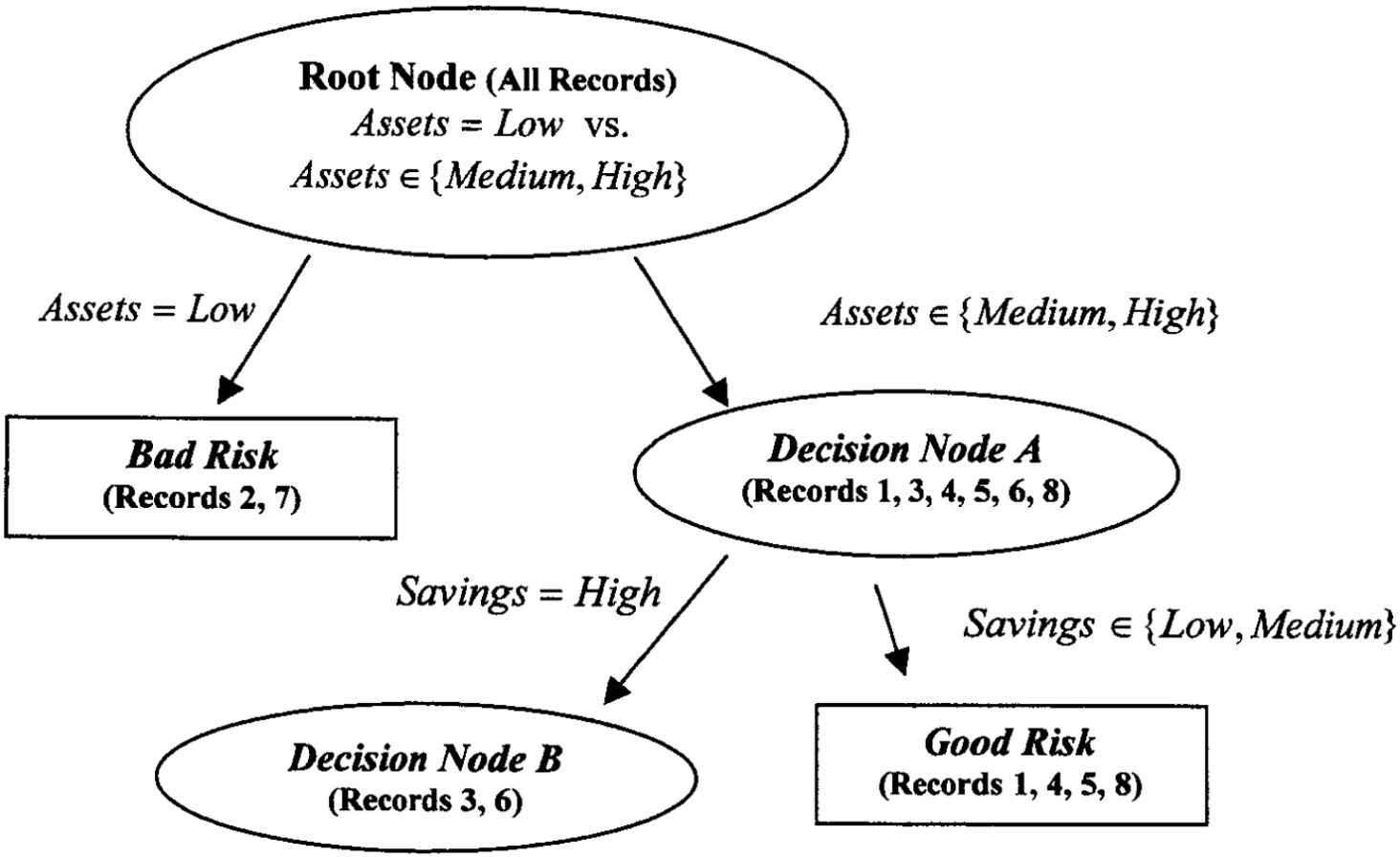

Figure 7: CART decision tree after decision node A split.

CART - Example - Third Split

Since decision node B is diverse, we again need to seek the optimal split. Only two records remain in this decision node, each with the same value for savings (high) and income (25). Therefore, the only possible split is assets = high versus assets = medium, providing us with the final form of the CART decision tree for this example, in Figure 8.

CART - Example - Third Split

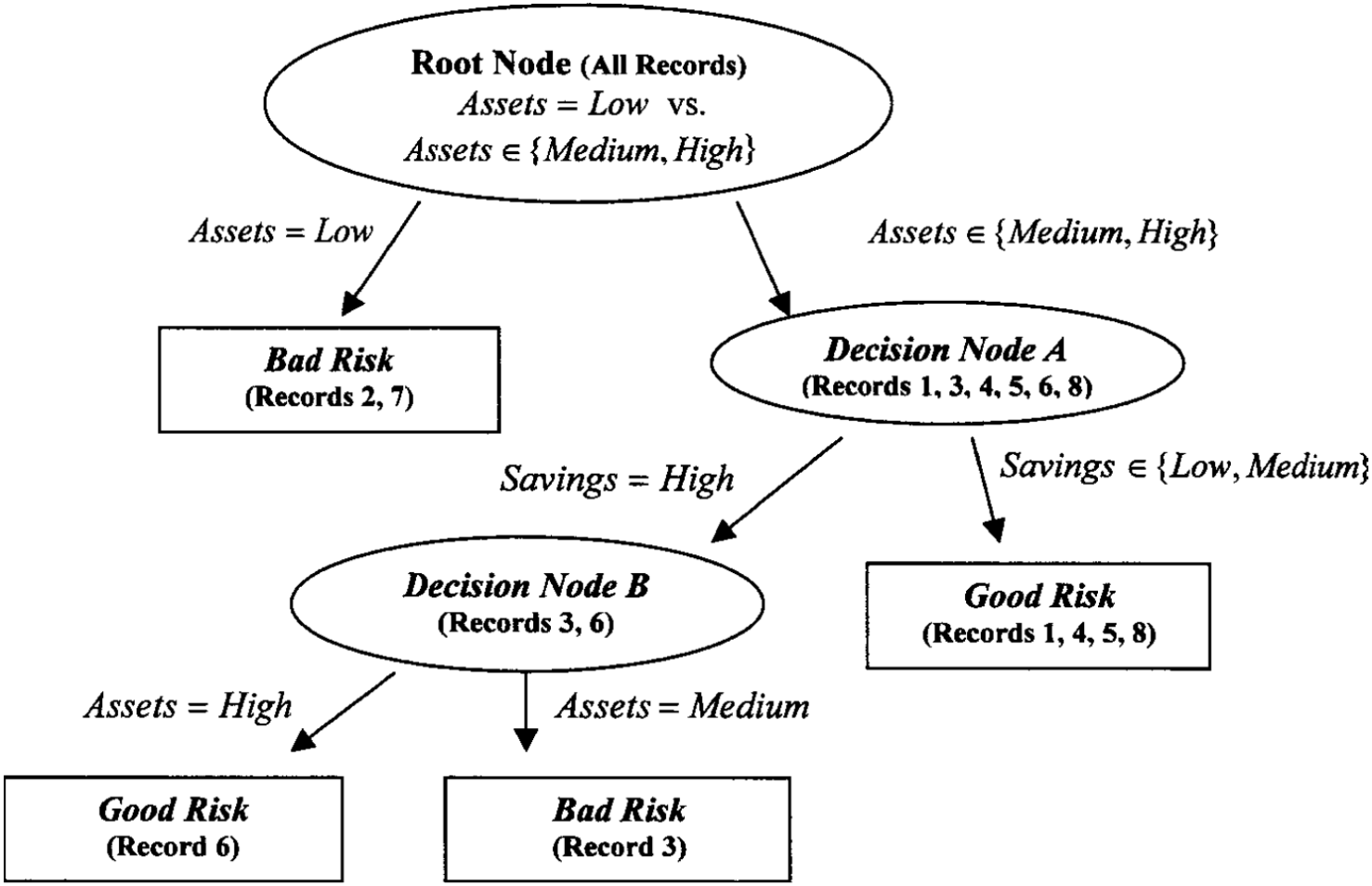

Figure 8: CART decision tree, fully grown form.

CART - Classification Error

Let us leave aside this example now, and consider how CART would operate on an arbitrary data set.

In general, CART would recursively proceed to visit each remaining decision node and apply the procedure above to find the optimal split at each node. Eventually, no decision nodes remain, and the "**full tree**" has been grown.

However, as we have seen in Table 1, not all leaf nodes are necessarily homogeneous, which leads to a certain level of *classification error*.

CART - Classification Error - Example

For example, suppose that since we cannot further partition the records in Table 1, we classify the records contained in this leaf node as bad credit risk.

Then the probability that a randomly chosen record from this leaf node would be classified correctly is 0.6, since three of the five records (60%) are actually classified as bad credit risks. Hence, our classification error rate for this particular leaf would be 0.4 or 40%, since two of the five records are actually classified as good credit risks.

CART would then calculate the error rate for the entire decision tree to be the weighted average of the individual leaf error rates, with the weights equal to the proportion of records in each leaf.

CART - Pruning Decision Trees

To avoid memorizing the training set, the CART algorithm needs to begin pruning nodes and branches that would otherwise reduce the generalizability of the classification results. Even though the fully grown tree has the lowest error rate on the training set, the resulting model may be too complex, resulting in over-fitting. As each decision node is grown, the subset of records available for analysis becomes smaller and less representative of the overall population. Pruning the tree will increase the generalizability of the results. How the CART algorithm performs tree pruning is explained in Breiman et al. [1, p. 66].

Essentially, an adjusted overall error rate is found that penalizes the decision tree for having too many leaf nodes and thus too much complexity.

Measures for selecting the Best Split

There are many measures that can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting.

Let $p(i|t)$ denote the fraction of records belonging to class i at a given node t. We sometimes omit the reference to node t and express the fraction as p_i. In a two class problem, the class distribution at any node can be written as (p_0, p_1), where $p_1 = 1 - p_0$. To illustrate consider the test conditions shown in Figure 9.

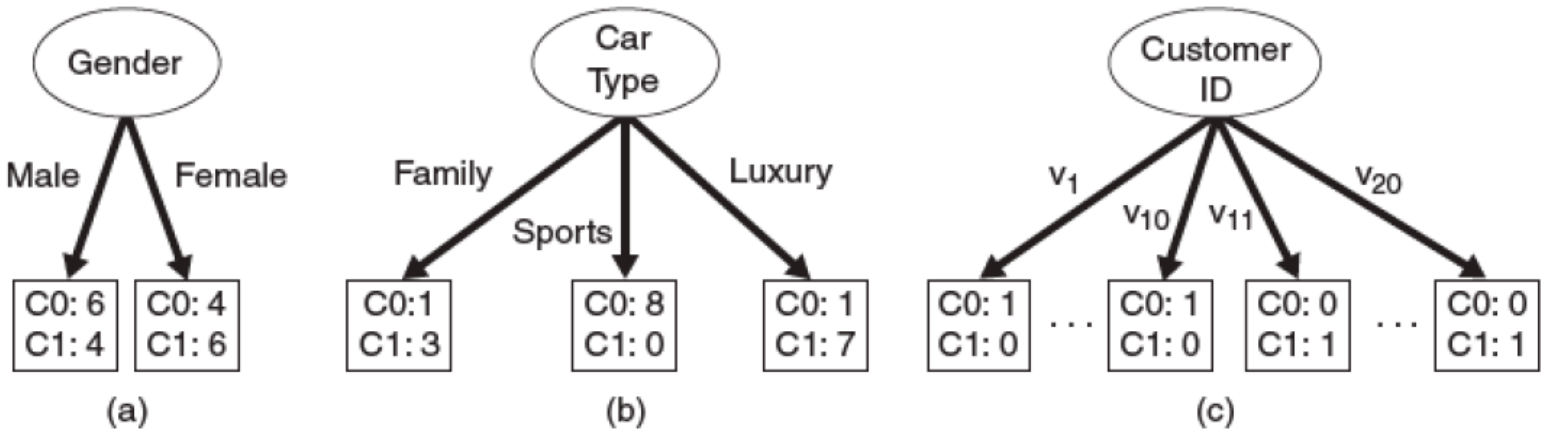

Figure 9: Multi-way vs binary splits

Measures for selecting the Best Split

The class distribution before splitting is $(0.5, 0.5)$ because there is an equal number of records from each class. If we split the data using the Gender attribute, then the class distributions of the child nodes are (0.6,0.4) and (0.4,0.6) respectively. Although the classes are no longer evenly distributed, the child nodes still contain records from both classes. Splitting on the second attribute, Car Type, will result in purer partitions.

Measures for selecting the Best Split

The measures developed for selecting the best split are often based on the degree of impurity of the child nodes. The smaller the degree of impurity, the more skewed the class distribution. For example, a node with class distribution (0,1) has zero impurity wheres a node with uniform class distribution has the highest impurity. Examples of impurity measures include:

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t), \tag{2}$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2, \tag{3}$$

$$\text{Classification Error}(t) = 1 - \max_i [p(i|t)], \tag{4}$$

where, c is the number of classes and $0 \log_2 0 = 0$ in entropy calculations.

Comparison among measures for binary split

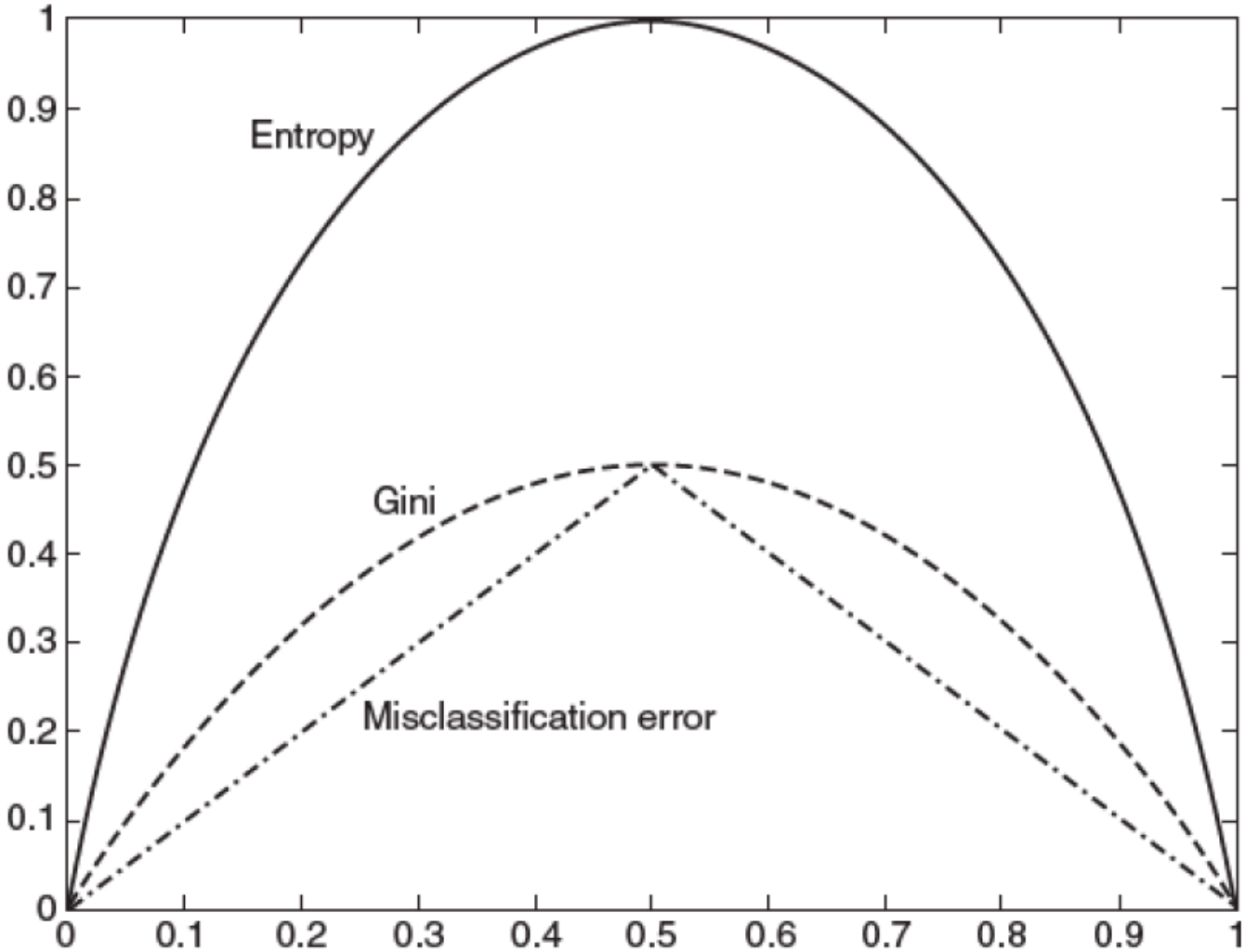

Figure 10: Comparison among the impurity measures for binary classification problems

Example of impurity measures for binary split

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

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

Figure 11: Example score for each impurity measure

C4.5 algorithm

The C4.5 algorithm is Quinlan's extension of his own ID3 algorithm for generating decision trees [3]. Just as with CART, the C4.5 algorithm recursively visits each decision node, selecting the optimal split, until no further splits are possible. However, there are interesting differences between CART and C4.5:

- Unlike CART, the C4.5 algorithm is not restricted to binary splits. Whereas CART always produces a binary tree, C4.5 produces a tree of more variable shape.
- For categorical attributes, C4.5 by default produces a separate branch for each value of the categorical attribute. This may result in more "congested" than desired, since some values may have low frequency or may naturally be associated with other values.
- The C4.5 method for measuring node homogeneity is quite different from the CART method and is examined in detail below.

C4.5 algorithm - Information Gain

The C4.5 algorithm uses the concept of information gain or entropy reduction to select the optimal split.

C4.5 uses this concept of entropy as follows. Suppose that we have a candidate split S, which partitions the training data set T into several subsets, $T_1, T_2, \ldots, T_k$. The mean information requirement can then be calculated as the weighted sum of the entropies for the individual subsets, as follows:

$$H_s(T) = \sum_{i=1}^{k} P_i H_s(T_i) \tag{5}$$

where P_i represents the proportion of records in subset i . We may then define our information gain to be $gain(S) = H(T) - H_S(T)$, that is, the increase in information produced by partitioning the training data T according to this candidate split S. At each decision node, C4.5 chooses the optimal split to be the split that has the greatest information gain, gain(S).

C4.5 algorithm - Example

To illustrate the C4.5 algorithm at work, let us return to the data set in Figure 2 and apply the C4.5 algorithm to build a decision tree for classifying credit risk, just as we did earlier using CART. Once again, we are at the root node and are considering all possible splits using all the data (Figure 12).

Candidate Split	Child Nodes		
1	$Savings = low$	$Savings = medium$	$Savings = high$
2	$Assets = low$	$Assets = medium$	$Assets = high$
3	$Income \leq \$25{,}000$		$Income > \$25{,}000$
4	$Income \leq \$50{,}000$		$Income > \$50{,}000$
5	$Income \leq \$75{,}000$		$Income > \$75{,}000$

Figure 12: Candidate Splits at Root Node for C4.5 Algorithm

C4.5 algorithm - Example - Before Split

Now, because five of the eight records are classified as good credit risk, with the remaining three records classified as bad credit risk, the entropy before splitting is:

$$H(T) = -\sum_j p_j \log_2(p_j) = -\frac{5}{8}\log_2\frac{5}{8} - \frac{3}{8}\log_2\frac{3}{8} = 0.9544$$

We shall compare the entropy of each candidate split against this $H(T) = 0.9544$, to see which split results in the greatest reduction in entropy (or gain in information).

C4.5 algorithm - Example - Split 1(High Savings)

For candidate split 1 (savings), two of the records have high savings, three of the records have medium savings, and three of the records have low savings, so we have: $P_{high} = \frac{2}{8}$, $P_{medium} = \frac{3}{8}$, $P_{low} = \frac{3}{8}$.

Of the records with high savings, one is a good credit risk and one is bad, giving a probability of 0.5 of choosing the record with a good credit risk.

Thus, the entropy for high savings is $-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$, which is similar to the flip of a fair coin. All three of the records with medium savings are good credit risks, so that the entropy for medium is $-\frac{3}{3}\log_2(\frac{3}{3}) - \frac{0}{3}\log_2(\frac{0}{3}) = 1$, where by convention we define $\log_2(0) = 0$.

C4.5 algorithm - Example - Split 1(Medium Savings)

In engineering applications, information is analogous to signal, and entropy is analogous to noise, so it makes sense that the entropy for medium savings is zero, since the signal is crystal clear and there is no noise: If the customer has medium savings, he or she is a good credit risk, with 100% confidence.

The amount of information required to transmit the credit rating of these customers is zero, as long as we know that they have medium savings.

C4.5 algorithm - Example - Split 1(Low Savings)

One of the records with low savings is a good credit risk, and two records with low savings are bad credit risks, giving us our entropy for low credit risk as $-\frac{1}{3}\log_2(\frac{1}{3}) - \frac{2}{3}\log_2(\frac{2}{3}) = 0.9183$.

C4.5 algorithm - Example - Information Gain

We combine the entropies of these three subsets, using equation (5) and the proportions of the subsets P_i , so that $H_{savings}(T) = \frac{2}{8}(1) + \frac{3}{8}(0) + \frac{3}{8}(0.9183) = 0.5944$.

Then the **information gain** represented by the split on the savings attribute is calculated as $H(T) - H_{savings}(T) = 0.9544 - 0.5944 = 0.36$ bits.

C4.5 algorithm - Example - Interpretation after Split 1

How are we to interpret these measures?

First, $H(T) = 0.9544$ means that, on average, one would need 0.9544 bit (0's or 1's) to transmit the credit risk of the eight customers in the data set. Now, $H_{savings}(T) = 0.5944$ means that the partitioning of the customers into three subsets has lowered the average bit requirement for transmitting the credit risk status of the customers to 0.5944 bits.

Lower entropy is good.

This entropy reduction can be viewed as information gain, so that we have gained on average $H(T) - Hsavings(T) = 0.9544 - 0.5944 = 0.36$ bits of information by using the savings partition.

We will compare this to the information gained by the other candidate splits, and **choose the split with the largest information gain as the optimal split** for the root node.

C4.5 algorithm - Example - Information Gain for Split 1

Candidate Split	Child Nodes	Information Gain (Entropy Reduction)
1	$Savings = low$ $Savings = medium$ $Savings = high$	0.36 bits
2	$Assets = low$ $Assets = medium$ $Assets = high$	0.5487 bits
3	$Income \leq \$25{,}000$ $Income > \$25{,}000$	0.1588 bits
4	$Income \leq \$50{,}000$ $Income > \$50{,}000$	0.3475 bits
5	$Income \leq \$75{,}000$ $Income > \$75{,}000$	0.0923 bits

Figure 13: Information Gain for Each Candidate Split at the Root Node

C4.5 algorithm - Example - Split 1

Figure 13 summarizes the information gain for each candidate split at the root node. Candidate split 2, assets, has the largest information gain, and so is chosen for the initial split by the C4.5 algorithm. Note that this choice for an optimal split concurs with the partition preferred by CART, which split on assets = low versus assets $\in$ {medium, high}. The partial decision tree resulting from C4.5's initial split is shown in Figure 14.

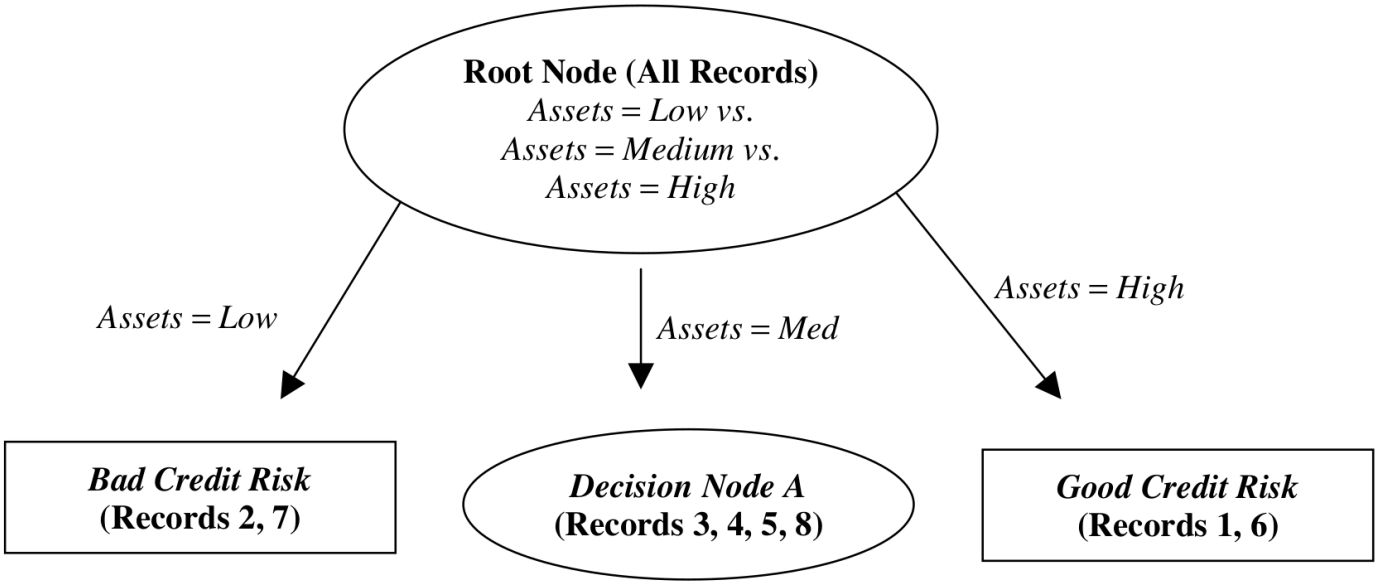

Figure 14: C4.5 concurs with CART in choosing assets for the initial partition.

C4.5 algorithm - Example - Before Split 2

Customer	Savings	Assets	Income ($1000s)	Credit Risk
3	High	Medium	25	Bad
4	Medium	Medium	50	Good
5	Low	Medium	100	Good
8	Medium	Medium	75	Good

Figure 15: Records Available at Decision Node A for Classifying Credit Risk

We proceed to determine the optimal split for decision node A, containing records 3, 4, 5, and 8, as indicated in Figure 15. Because three of the four records are classified as good credit risks, with the remaining record classified as a bad credit risk, the entropy before splitting is

$$H(A) = -\sum_j p_j \log_2(p_j) = -\frac{3}{4}\log_2(\frac{3}{4}) - \frac{1}{4}\log_2(\frac{1}{4}) = 0.8113$$

C4.5 algorithm - Example - Split 2

Candidate Split	Child Nodes		
1	$Savings = low$	$Savings = medium$	$Savings = high$
3	$Income \leq \$25{,}000$		$Income > \$25{,}000$
4	$Income \leq \$50{,}000$		$Income > \$50{,}000$
5	$Income \leq \$75{,}000$		$Income > \$75{,}000$

Figure 16: Candidate Splits at Decision Node A

Figure 17 shows the form of the decision tree after the savings split. Note that this is the fully grown form, since all nodes are now leaf nodes, and C4.5 will grow no further nodes. Comparing the C4.5 tree in Figure 17 with the CART tree in Figure 8, we see that the C4.5 tree provides a greater breadth, while the CART tree is one level deeper. Both algorithms concur that assets is the most important variable (the root split) and that savings is also important. Finally, once the decision tree is fully grown, C4.5 engages in pessimistic postpruning. Interested readers may consult [4, p. 153].

C4.5 algorithm - Example - Split 2

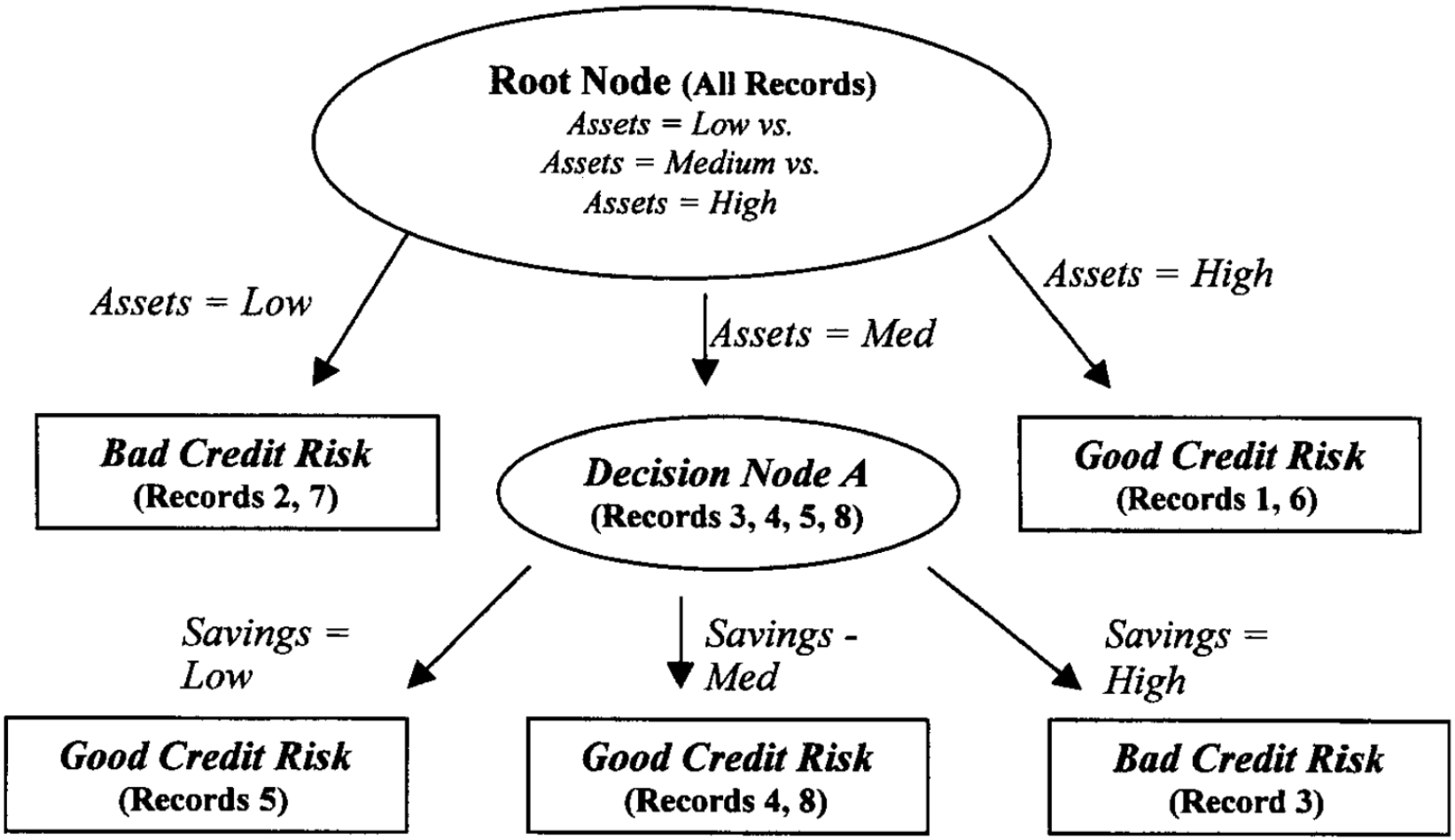

Figure 17: C4.5 Decision tree: fully grown form.

Decision Rules

One of the most attractive aspects of decision trees lies in their interpretability, especially with respect to the construction of decision rules. Decision rules can be constructed from a decision tree simply by traversing any given path from the root node to any leaf. The complete set of decision rules generated by a decision tree is equivalent (for classification purposes) to the decision tree itself. For example, from the decision tree in Figure 17, we may construct the decision rules given in Table 18.

Antecedent	Consequent	Support	Confidence
If *assets* = *low*	then *bad credit risk.*	$\frac{2}{8}$	1.00
If *assets* = *high*	then *good credit risk.*	$\frac{2}{8}$	1.00
If *assets* = *medium* and *savings* = *low*	then *good credit risk.*	$\frac{1}{8}$	1.00
If *assets* = *medium* and *savings* = *medium*	then *good credit risk.*	$\frac{2}{8}$	1.00
If *assets* = *medium* and *savings* = *high*	then *bad credit risk.*	$\frac{1}{8}$	1.00

Figure 18: Decision Rules Generated from Decision Tree in Figure 17

Decision Rules - Support and Confidence

The **support** of the decision rule refers to the proportion of records in the data set that rest in that particular terminal leaf node.

The **confidence** of the rule refers to the proportion of records in the leaf node for which the decision rule is true. In this small example, all of our leaf nodes are pure, resulting in perfect confidence levels of $100\% = 1.00$.

In real-world examples, one cannot expect such high confidence levels.

Further Reading

- Chapter 6.1 of Data Mining - Practical Machine Learning Tools and Techniques, Second Edition - Ian H. Witten, Eibe Frank
- Chapter 6 of DISCOVERING KNOWLEDGE IN DATA - An Introduction to Data Mining - DANIEL T. LAROSE
- Chapter 4.3 of Introduction to Data Mining (Second Edition) - Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, 1st ed. Chapman and Hall/CRC, 1984.

[2] R. Kennedy, *Solving data mining problems through pattern recognition*. Upper Saddle River, N.J: Prentice Hall PTR, 1997.

[3] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[4] M. Kantardzic, *Data mining : concepts, models, methods, and algorithms*. Piscataway, NJ Hoboken, New Jersey: IEEE Press Wiley, 2020.

Thank you.
Any Questions?