Five-Day Faculty Development Programme (FDP)
On
"Artificial Intelligence and Data Science: Foundations, Pedagogy, Tools, and Emerging Research Trends"
15–19 December 2025
Organized by the Department of Artificial Intelligence, SVNIT Surat
for the Faculty Members of P. P. Savani University

Supervised Learning: Classification

Dr. Rahul Shrivastava (Assistant Professor, DoAI, SVNIT, Surat)

- 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. Most algorithms are memory resident, typically assuming a small data size.
- Classification has numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, and medical diagnosis.

What Is Classification?

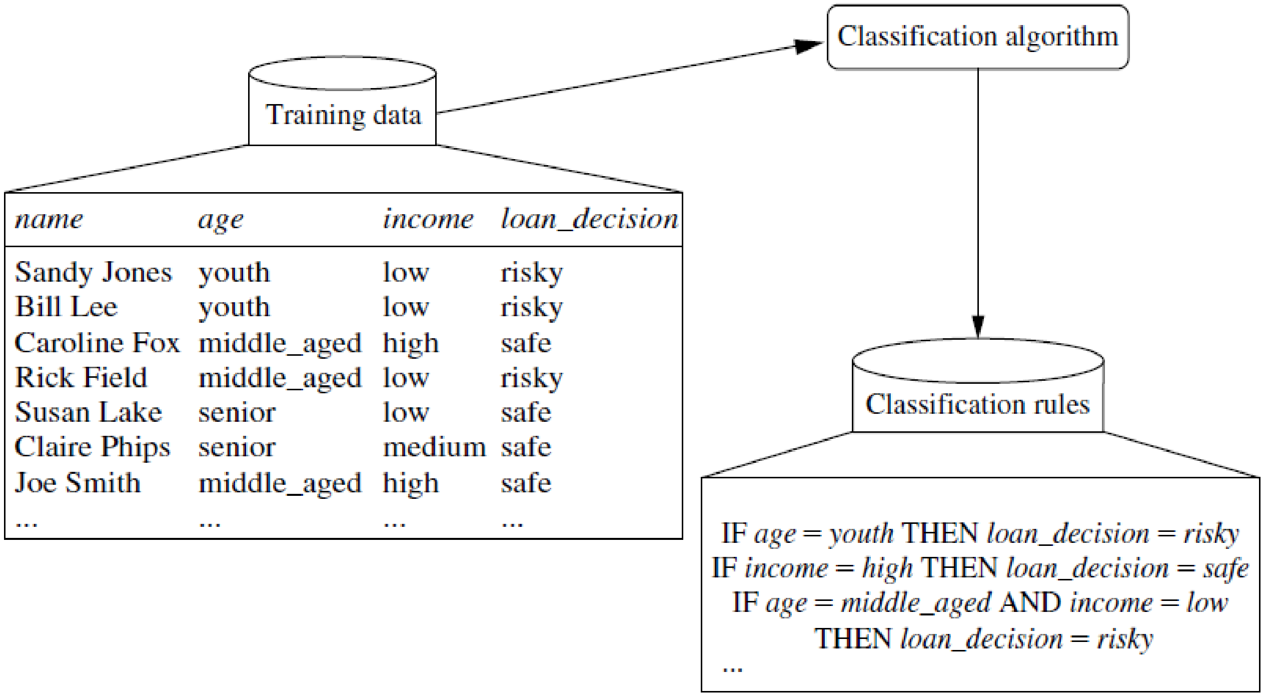
- A bank loans officer needs analysis of her data to learn which loan applicants are "safe" and which are "risky" for the bank.
- A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive.

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; 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).

(a) Learning: Training data are analyzed by a classification algorithm. Here, the class label attribute is loan decision, and the learned model or classifier is represented in the form of classification rules.

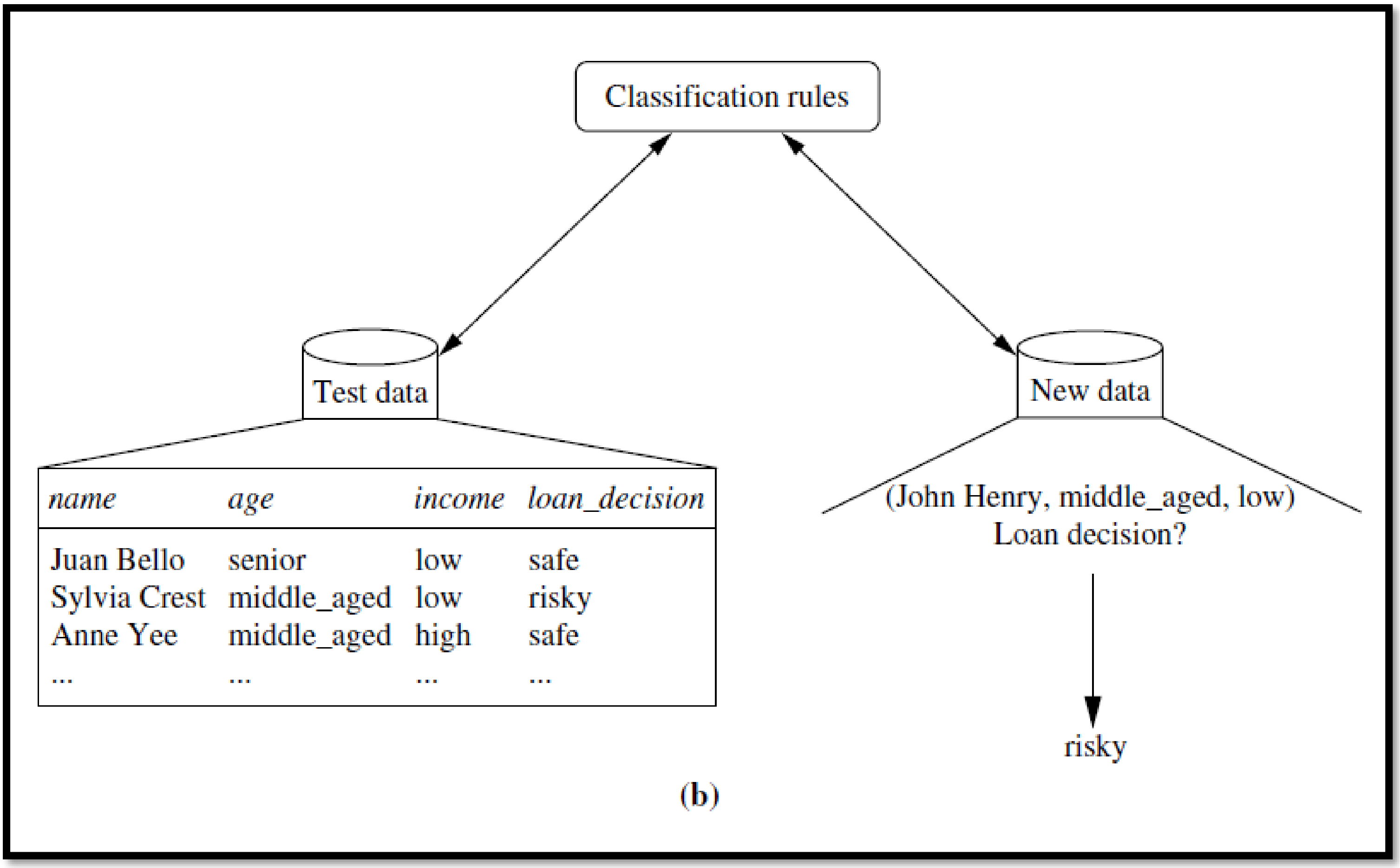


name	*age*	*income*	*loan_decision*
Sandy Jones	youth	low	risky
Bill Lee	youth	low	risky
Caroline Fox	middle_aged	high	safe
Rick Field	middle_aged	low	risky
Susan Lake	senior	low	safe
Claire Phips	senior	medium	safe
Joe Smith	middle_aged	high	safe
...	...	...	...

IF *age* = *youth* THEN *loan_decision* = *risky*
IF *income* = *high* THEN *loan_decision* = *safe*
IF *age* = *middle_aged* AND *income* = *low*
THEN *loan_decision* = *risky*
...

(a)

(b) Classification: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.



name	*age*	*income*	*loan_decision*
Juan Bello	senior	low	safe
Sylvia Crest	middle_aged	low	risky
Anne Yee	middle_aged	high	safe
...	...	...	...

(b)

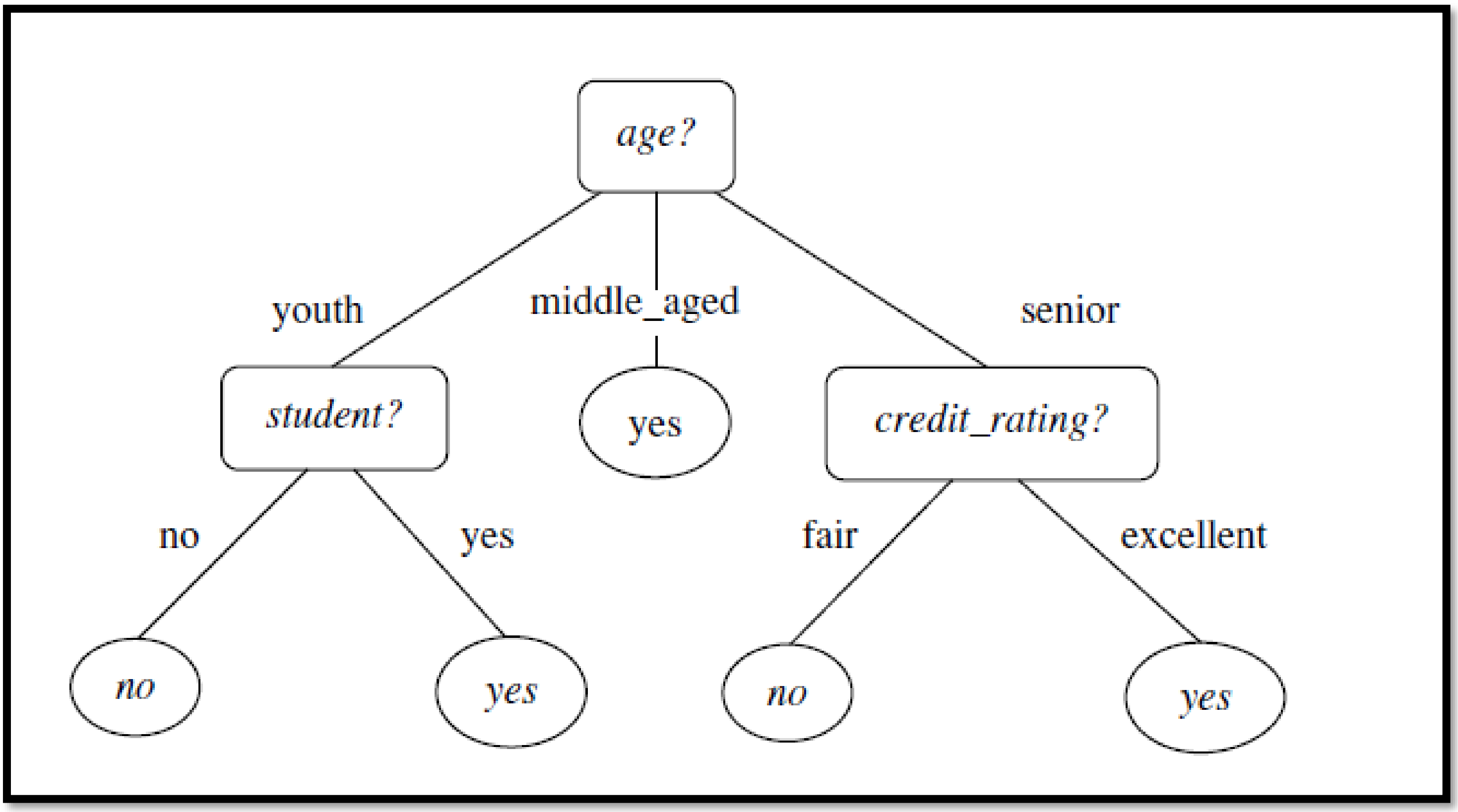
- Because the class label of each training tuple is provided, this step is also known as supervised learning (i.e., the learning of the classifier is “supervised” in that it is told to which class each training tuple belongs).
- It contrasts with unsupervised learning (or clustering), in which the class label of each training tuple is not known, and the number.
- This first step of the classification process can also be viewed as the learning of a mapping or function, *y=f(X)*, that can predict the associated class label y of a given tuple X.

“What about classification accuracy?”

- 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 decision tree for the concept buys_computer, indicating whether an customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either buys_computer= yes or buys computer= no). be learned may not be known in advance.

“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. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand.
- 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.

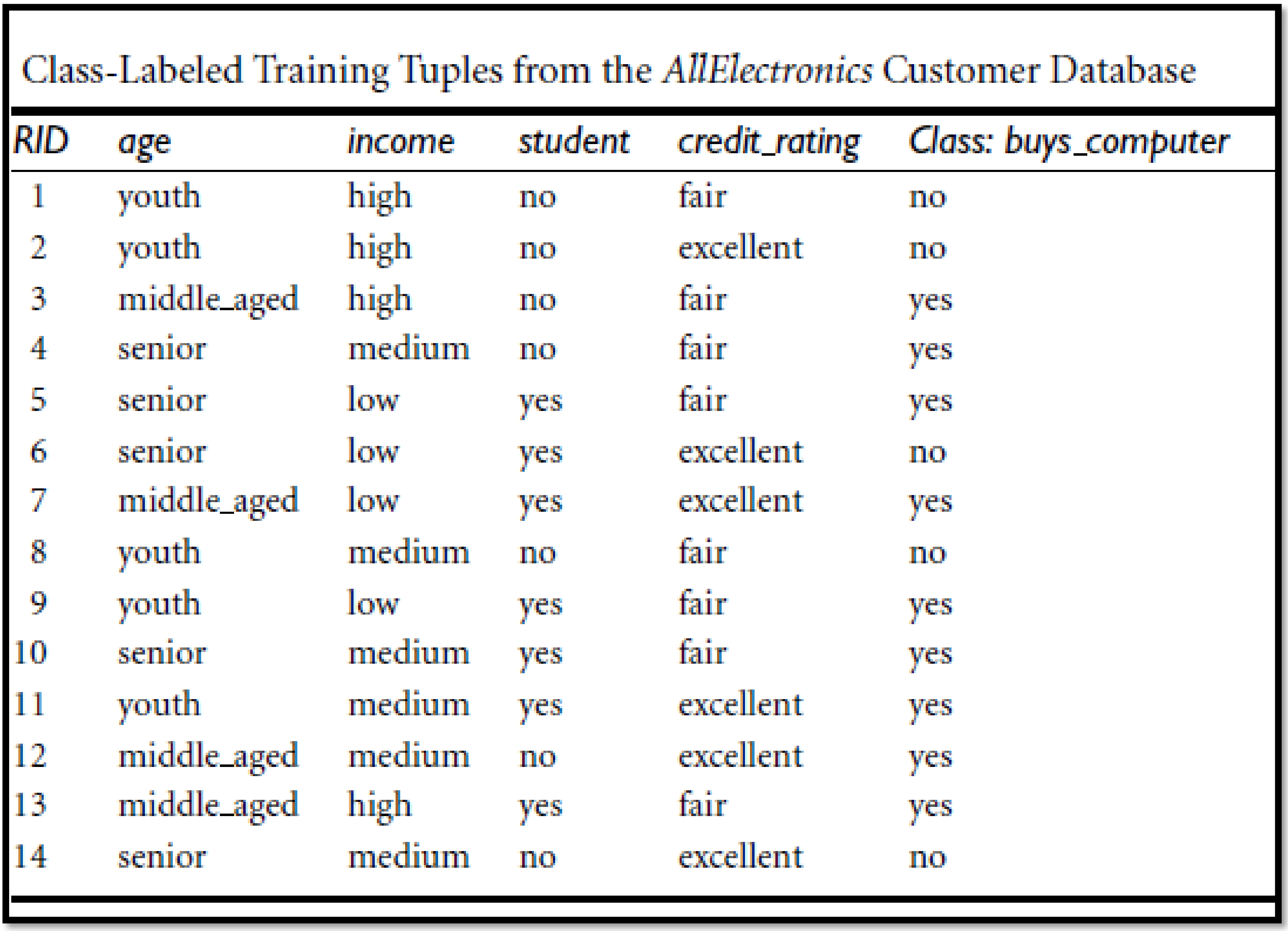
Decision trees are the basis of several commercial rule induction systems.

Decision Tree Induction

- **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.**
- **ID3 and CART were invented independently of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples.**

Decision Tree Algorithm

1. 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.

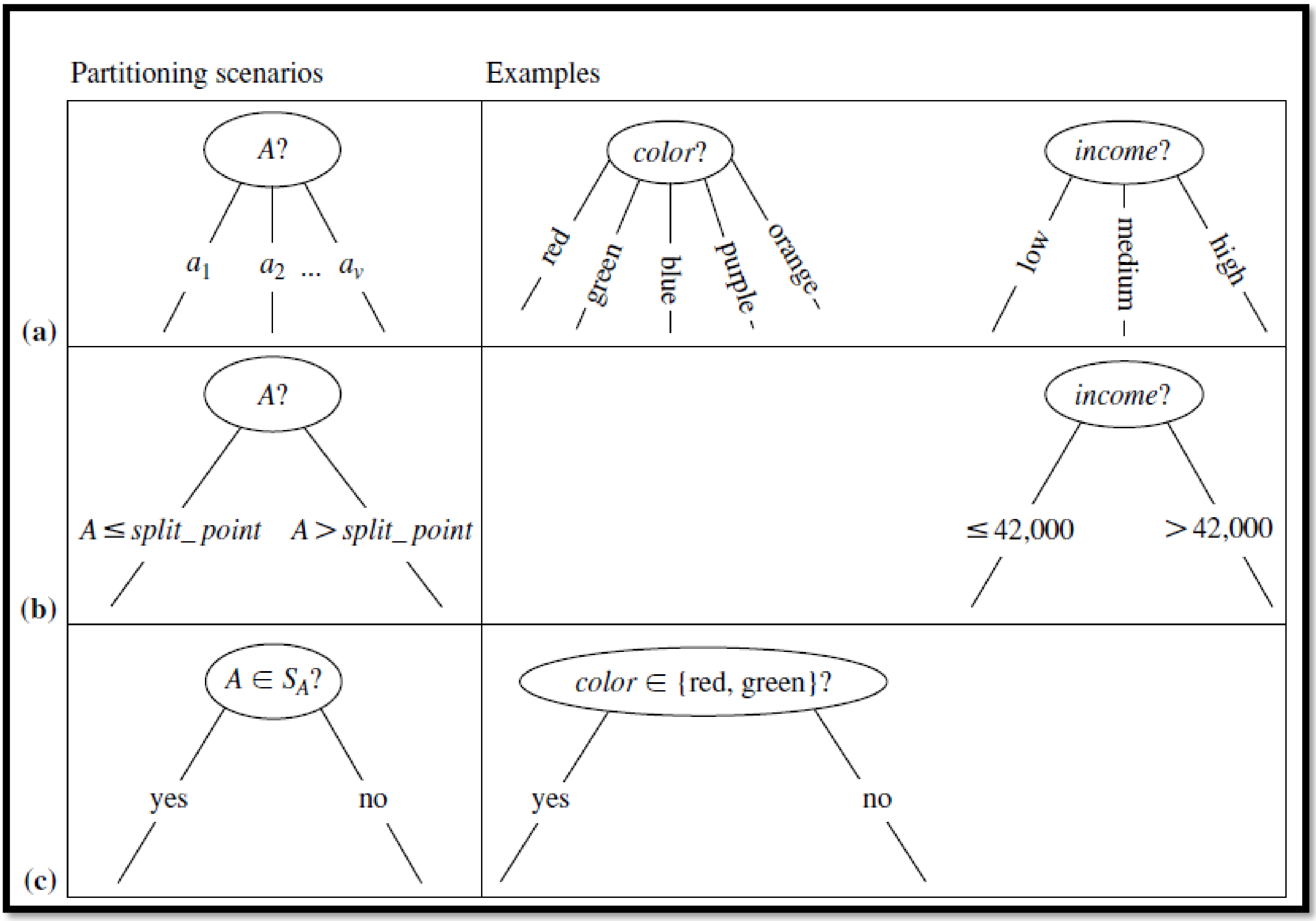


Class-Labeled Training Tuples from the *AllElectronics* Customer Database

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

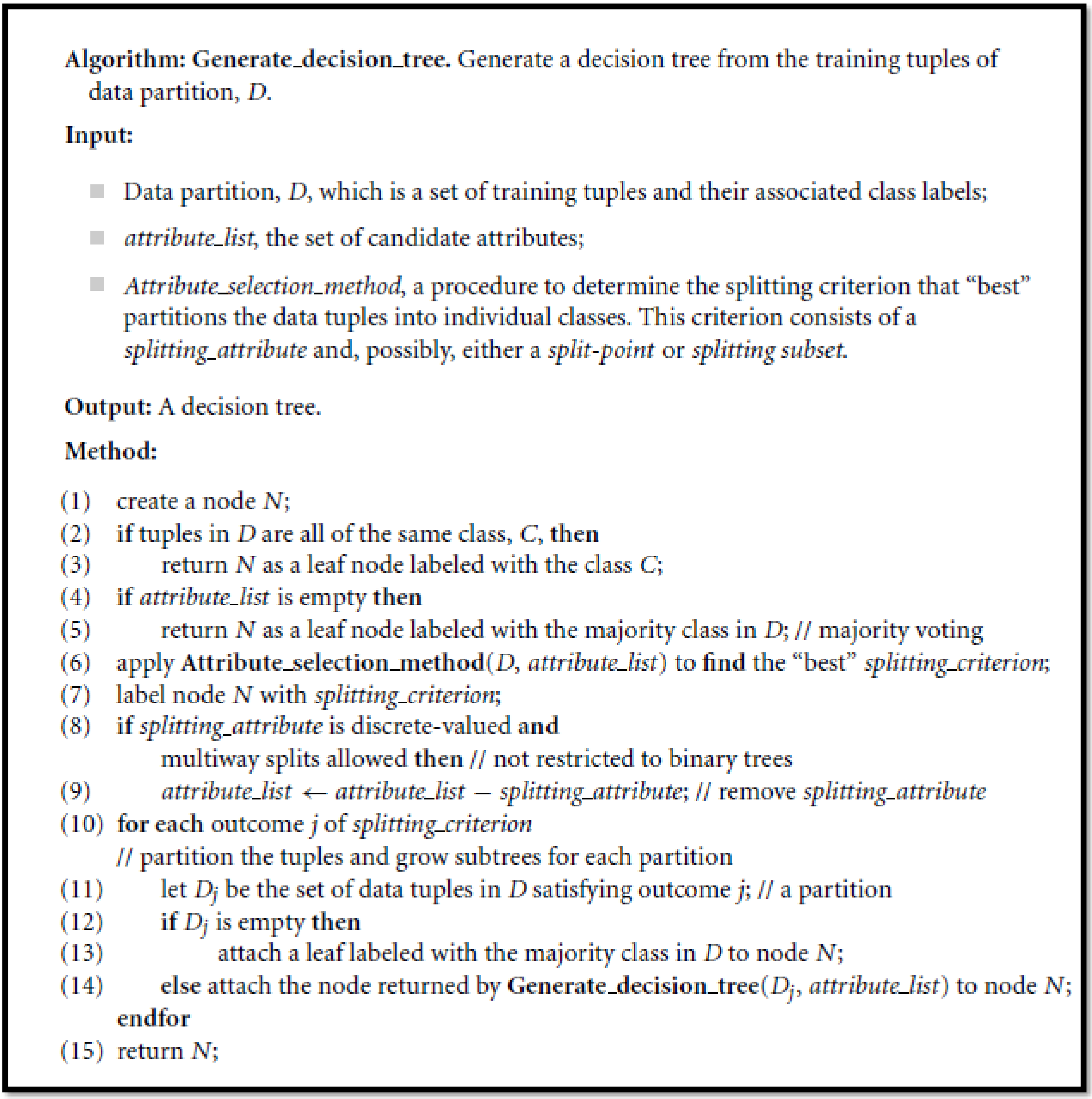
2. The tree starts as a single node, N, representing the training tuples in D.
3. If the tuples in D are all of the same class, then node N becomes a leaf and is labelled with that class.
4. Otherwise, the algorithm calls Attribute selection method to determine the splitting criterion. The splitting 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.
    a. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. A partition is pure if all the tuples in it belong to the same class.

5. The node N is labelled with the splitting criterion, which serves as a test at the node. A branch is grown from node N for each of the outcomes of the splitting criterion.



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, (c) If A is discrete-valued

6. **The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, Dj , of D.**
7. **The recursive partitioning stops only when any one of the following terminating conditions is true:**
    - **All the tuples in partition D (represented at node N) belong to the same class.**
    - **There are no remaining attributes on which the tuples may be further partitioned. In this case, majority voting is employed.**
    - **There are no tuples for a given branch.**



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:

```
create a node N;
if tuples in D are all of the same class, C, then
    return N as a leaf node labeled with the class C;
if attribute_list is empty then
    return N as a leaf node labeled with the majority class in D; // majority voting
apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
label node N with splitting_criterion;
if splitting_attribute is discrete-valued and
    multiway splits allowed then // not restricted to binary trees
    attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
for each outcome j of splitting_criterion
// partition the tuples and grow subtrees for each partition
    let D_j be the set of data tuples in D satisfying outcome j; // a partition
    if D_j is empty then
        attach a leaf labeled with the majority class in D to node N;
    else attach the node returned by Generate_decision_tree(D_j, attribute_list) to node N;
endfor
return N;
```

Attribute Selection Measures

An attribute selection measure is a heuristic for electing the splitting criterion that "best" separates a given data partition, D, of class-labelled training.

Individual 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.**
- **The expected information needed to classify a tuple in D is given by**

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

where p_i is the nonzero probability that an arbitrary tuple in D belongs to class Ci and is estimated by

$$\boxed{|C_{i,D}|/|D|.}$$

Info(D) is also known as the entropy of D.

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

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

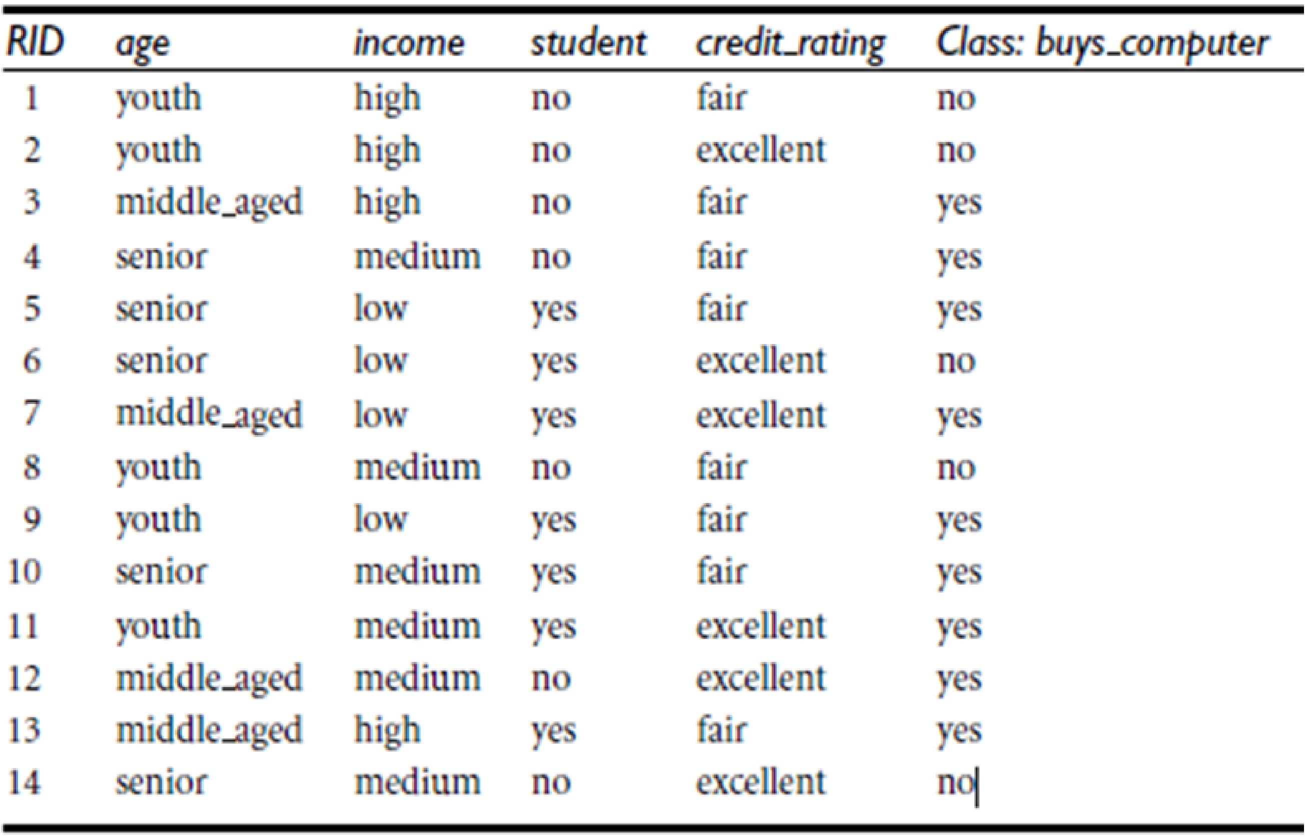
Where weight of the jth partition is

$$\frac{|D_j|}{|D|}$$

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,

$$\boxed{Gain(A) = Info(D) - Info_A(D)}$$

- In other words, Gain(A) tells us how much would be gained by branching on A. It is the expected reduction in the information requirement caused by knowing the value of A.
- The attribute A with the highest information gain, Gain.A/, is chosen as the splitting attribute at nodeN. 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 required to finish classifying the tuples is minimal



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

- Class P: buys_computer = “yes”=9
- Class N: buys_computer = “no”=5

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

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

The expected information needed to classify a tuple in D if the tuples are partitioned according to age is

Age	Pi	Ni	I(Pi,Ni)
Youth	2	3	.970
Middle_Aged	4	0	0
Senior	3	2	.970

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

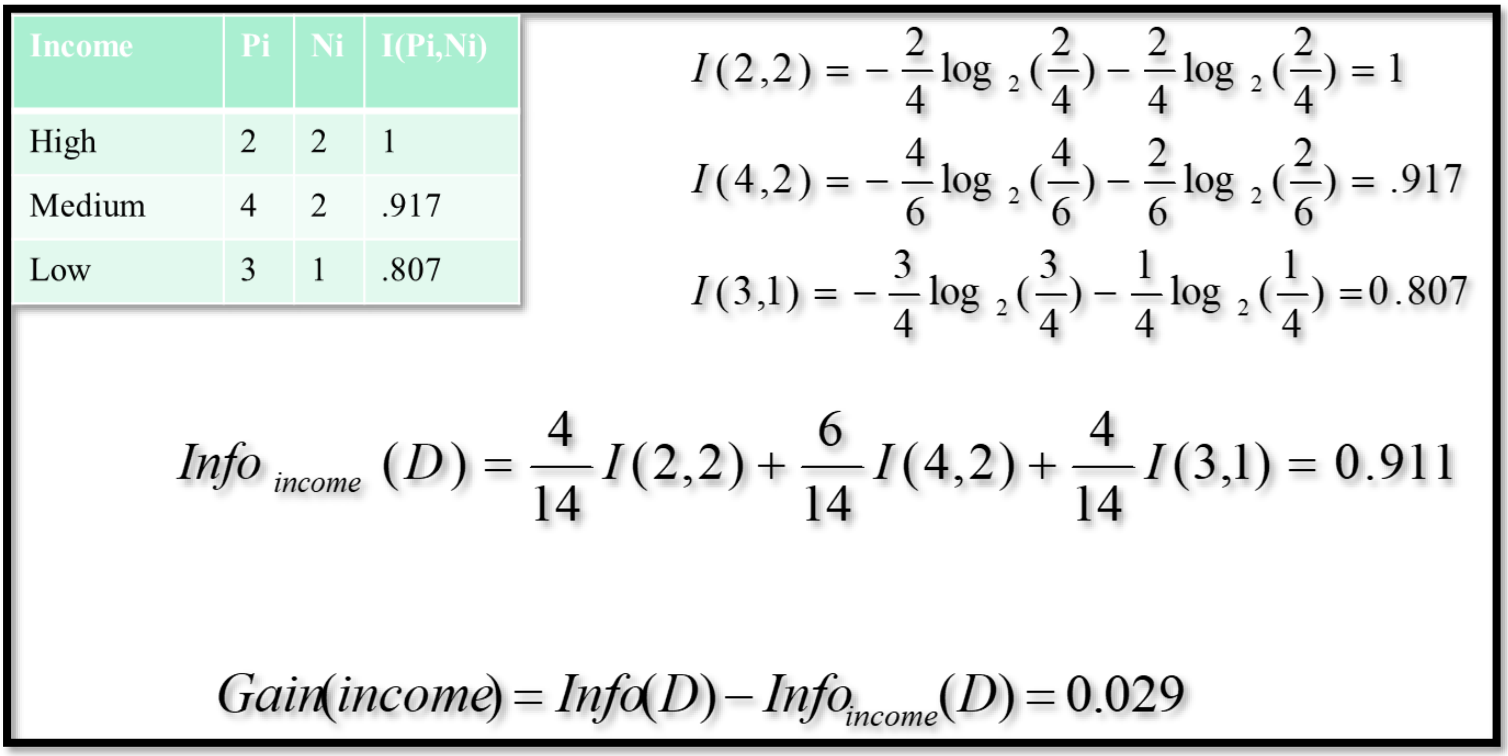
$$I(4,0) = -\frac{4}{4}\log_2(\frac{4}{4}) = 0$$

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

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

The expected information needed to classify a tuple in D if the tuples are partitioned according to Income is



Income	Pi	Ni	I(Pi,Ni)
High	2	2	1
Medium	4	2	.917
Low	3	1	.807

$$I(2,2) = -\frac{2}{4}\log_2(\frac{2}{4}) - \frac{2}{4}\log_2(\frac{2}{4}) = 1$$

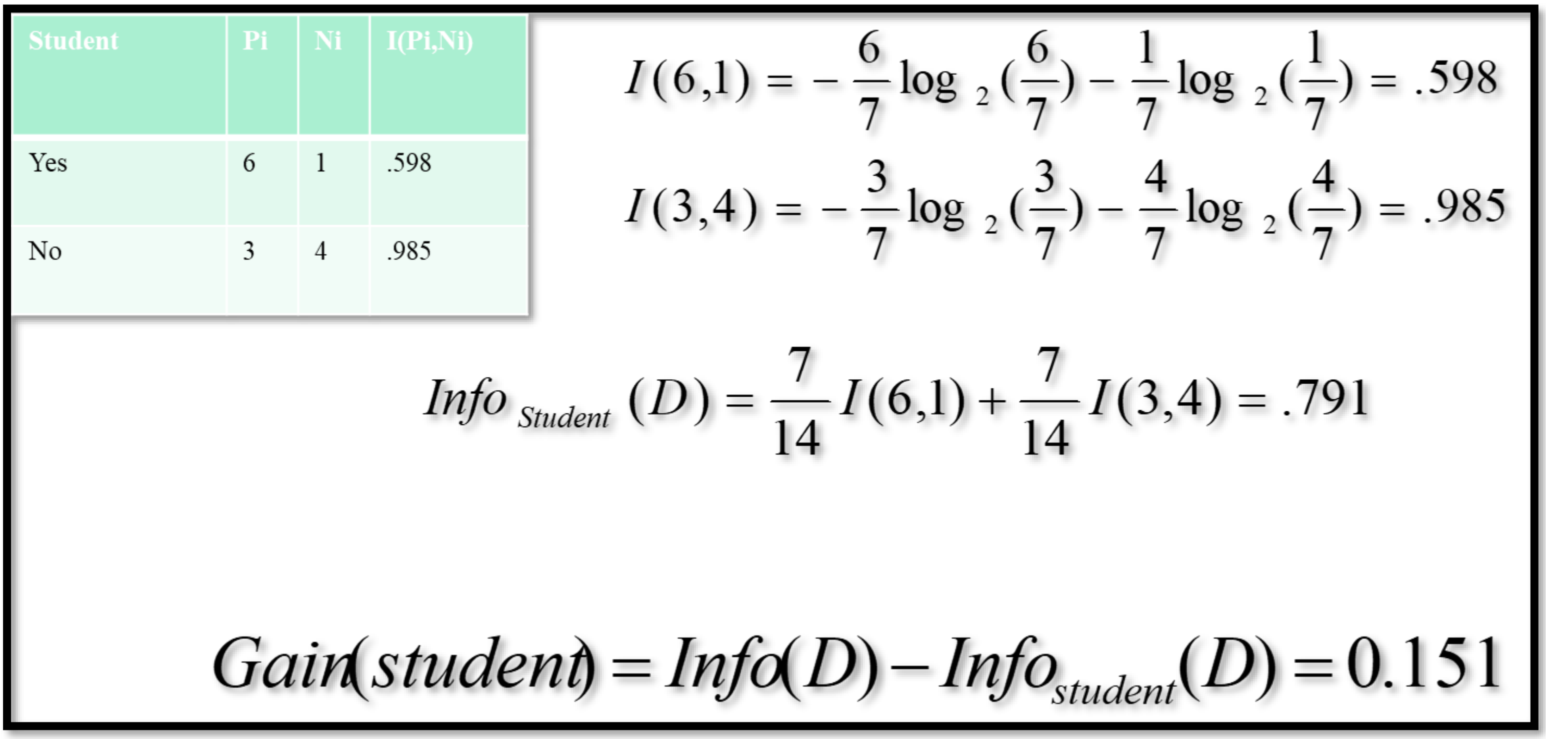
$$I(4,2) = -\frac{4}{6}\log_2(\frac{4}{6}) - \frac{2}{6}\log_2(\frac{2}{6}) = .917$$

$$I(3,1) = -\frac{3}{4}\log_2(\frac{3}{4}) - \frac{1}{4}\log_2(\frac{1}{4}) = 0.807$$

$$Info_{income}(D) = \frac{4}{14}I(2,2) + \frac{6}{14}I(4,2) + \frac{4}{14}I(3,1) = 0.911$$

$$Gain(income) = Info(D) - Info_{income}(D) = 0.029$$

The expected information needed to classify a tuple in D if the tuples are partitioned according to student is



Student	Pi	Ni	I(Pi,Ni)
Yes	6	1	.598
No	3	4	.985

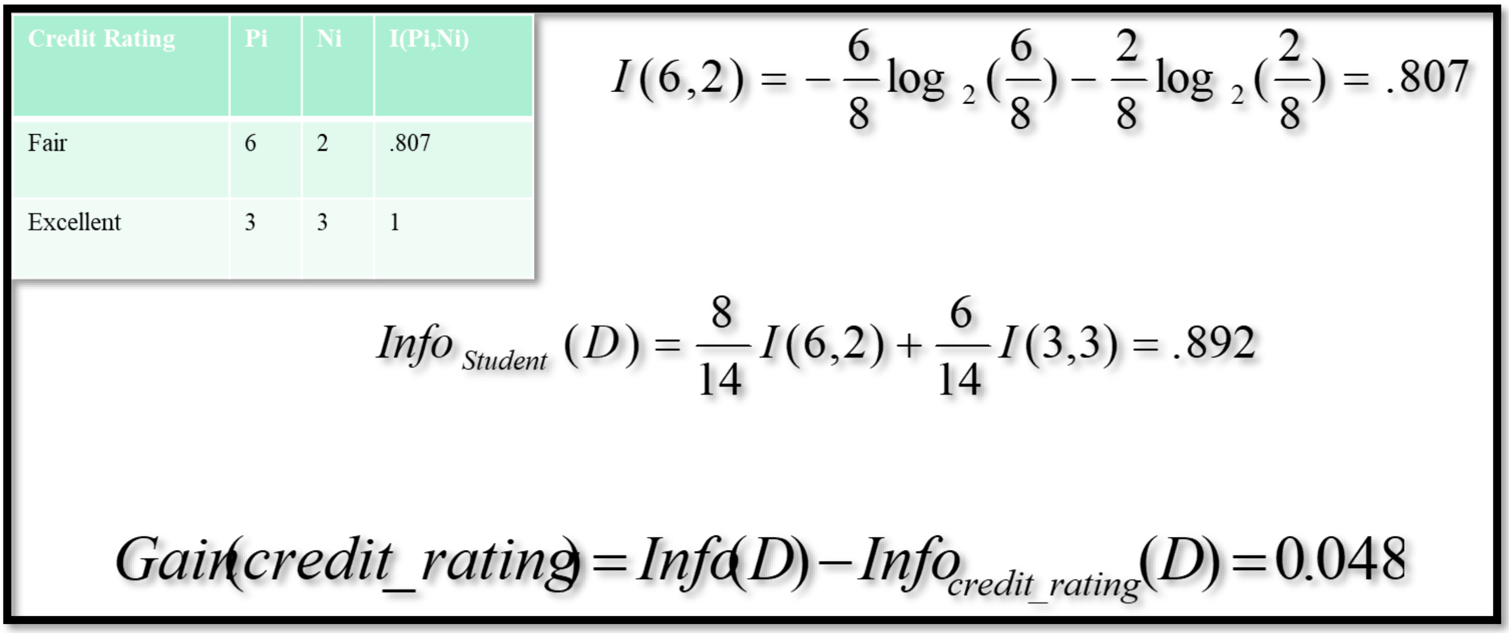
$$I(6,1) = -\frac{6}{7}\log_2(\frac{6}{7}) - \frac{1}{7}\log_2(\frac{1}{7}) = .598$$

$$I(3,4) = -\frac{3}{7}\log_2(\frac{3}{7}) - \frac{4}{7}\log_2(\frac{4}{7}) = .985$$

$$Info_{Student}(D) = \frac{7}{14}I(6,1) + \frac{7}{14}I(3,4) = .791$$

$$Gain(student) = Info(D) - Info_{student}(D) = 0.151$$

The expected information needed to classify a tuple in D if the tuples are partitioned according to Credit_Rating



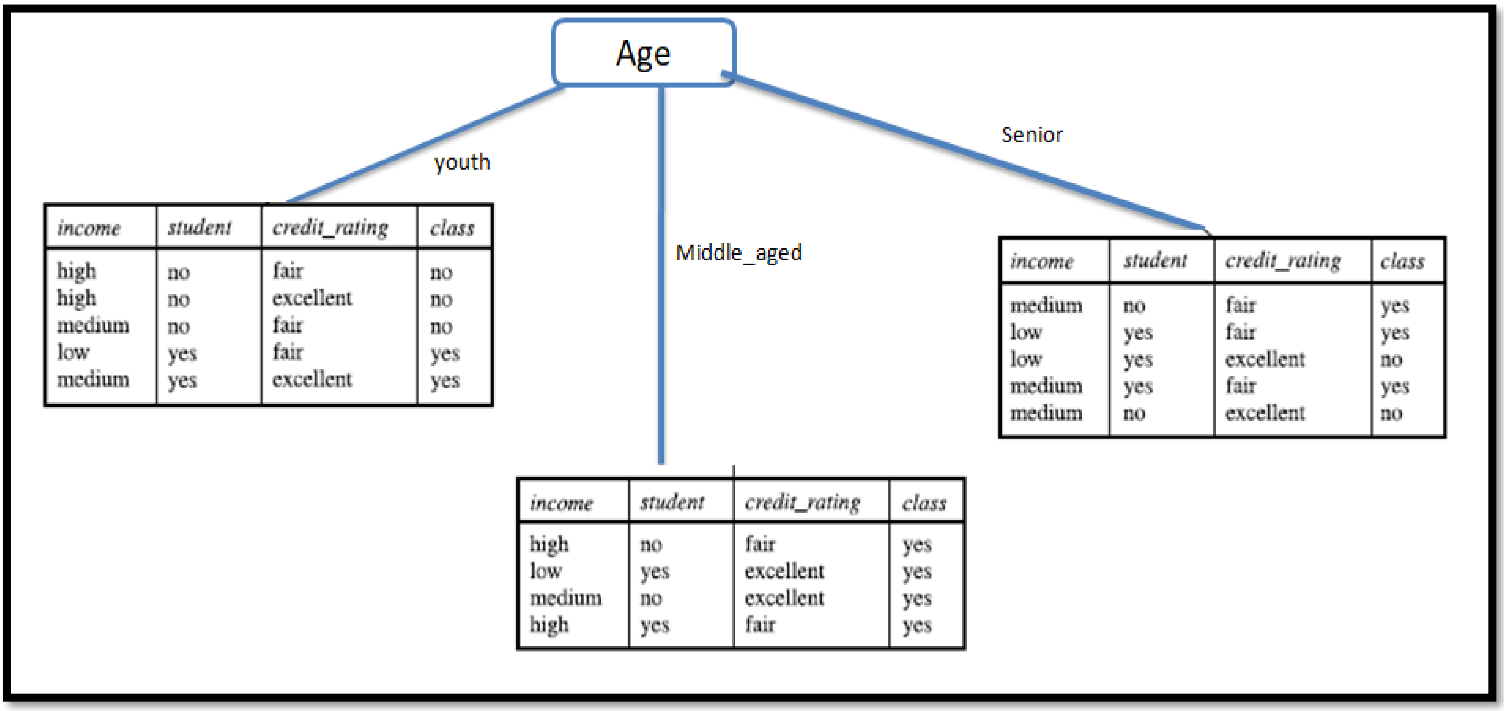
Credit Rating	Pi	Ni	I(Pi,Ni)
Fair	6	2	.807
Excellent	3	3	1

$$I(6,2) = -\frac{6}{8}\log_2(\frac{6}{8}) - \frac{2}{8}\log_2(\frac{2}{8}) = .807$$

$$Info_{Student}(D) = \frac{8}{14}I(6,2) + \frac{6}{14}I(3,3) = .892$$

$$Gain(credit\_rating) = Info(D) - Info_{credit\_rating}(D) = 0.048$$

‘age’ has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labelled with age, and branches are grown for each of the attribute’s values. Notice that the tuples falling into the partition for age D middle aged all belong to the same class. Because they all belong to class “yes,” a leaf should therefore be created at the end of this branch and labelled “yes.”

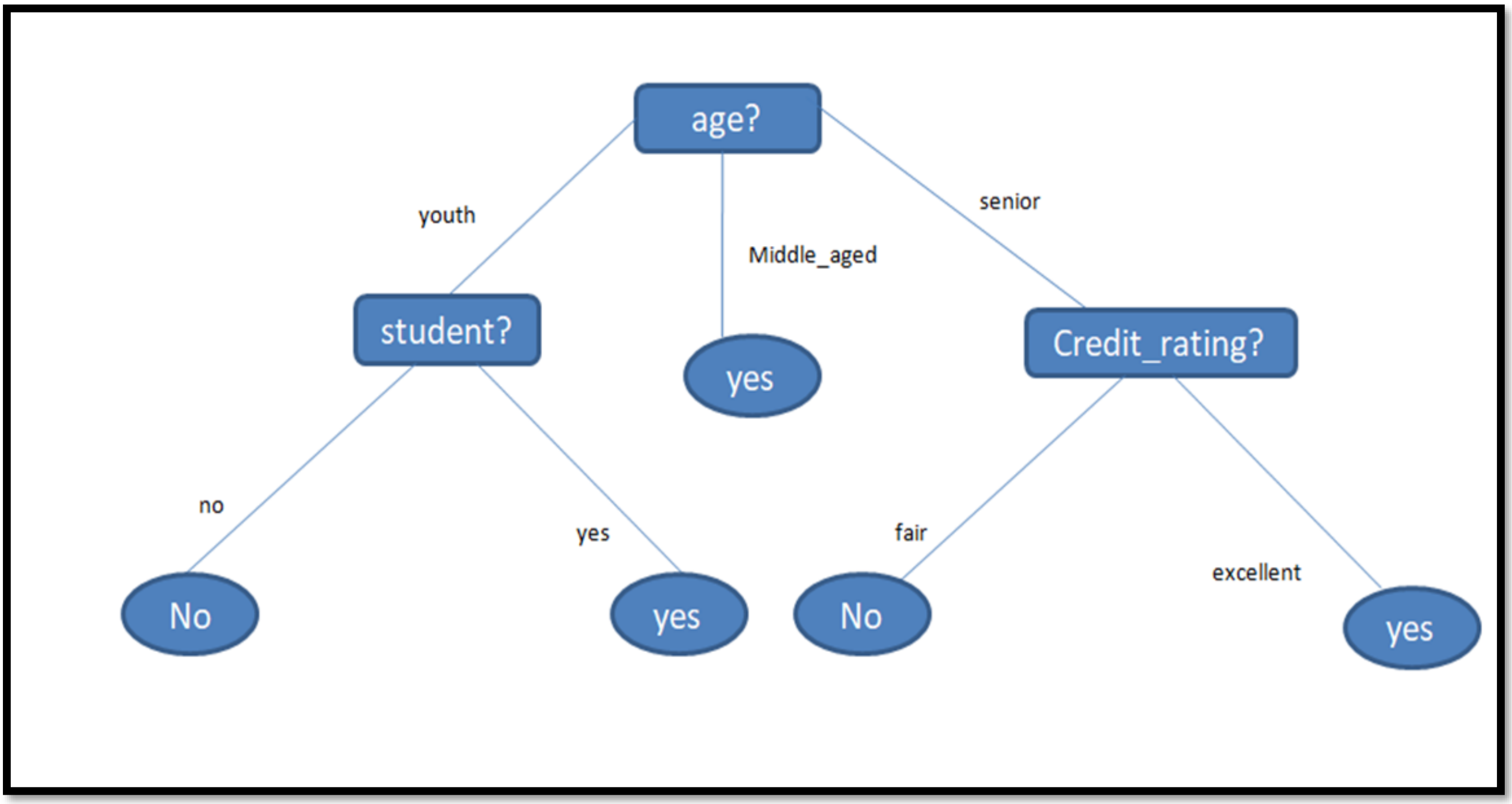


income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

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

Similar Calculation carried out for table D1 and Table D2 ,Student has the highest information gain among the attributes in table D1, and credit_rating has the highest information gain among the attributes in table D2 so the final decision tree can be constructed as follows:



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.
- 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:

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

- 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.

The gain ratio is defined as:

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

The attribute with the maximum gain ratio is selected as the splitting attribute.

Computation of gain ratio for the attribute income.

$$\begin{aligned} SplitInfo_{income}(D) &= -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) \\ &= 1.557. \end{aligned}$$

From earlier Example, 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

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D1 and D2, the Gini index of D given that partitioning is:

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

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is:

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

Gini index to compute the impurity of D:

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

Let's start with the attribute income and consider each of the possible splitting subsets. Consider the subset

{low, medium}

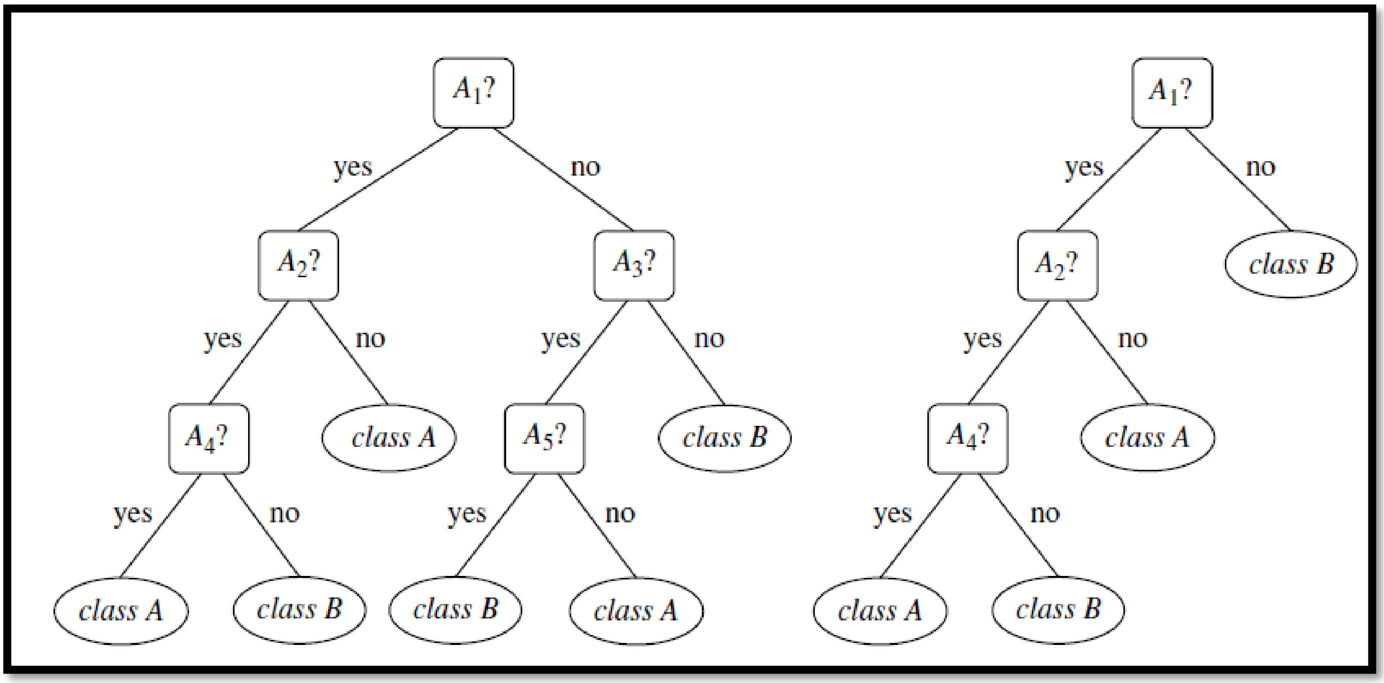
$$\begin{aligned}
&Gini_{income \in \{low,medium\}}(D)\\
&\quad = \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2)\\
&\quad = \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)\\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

Similarly, the Gini index values for splits on the remaining subsets are obtained. The attribute age and splitting subset {youth, senior} therefore give the minimum Gini index and results in the maximum reduction in impurity of the tuples.

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.

- **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 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.**

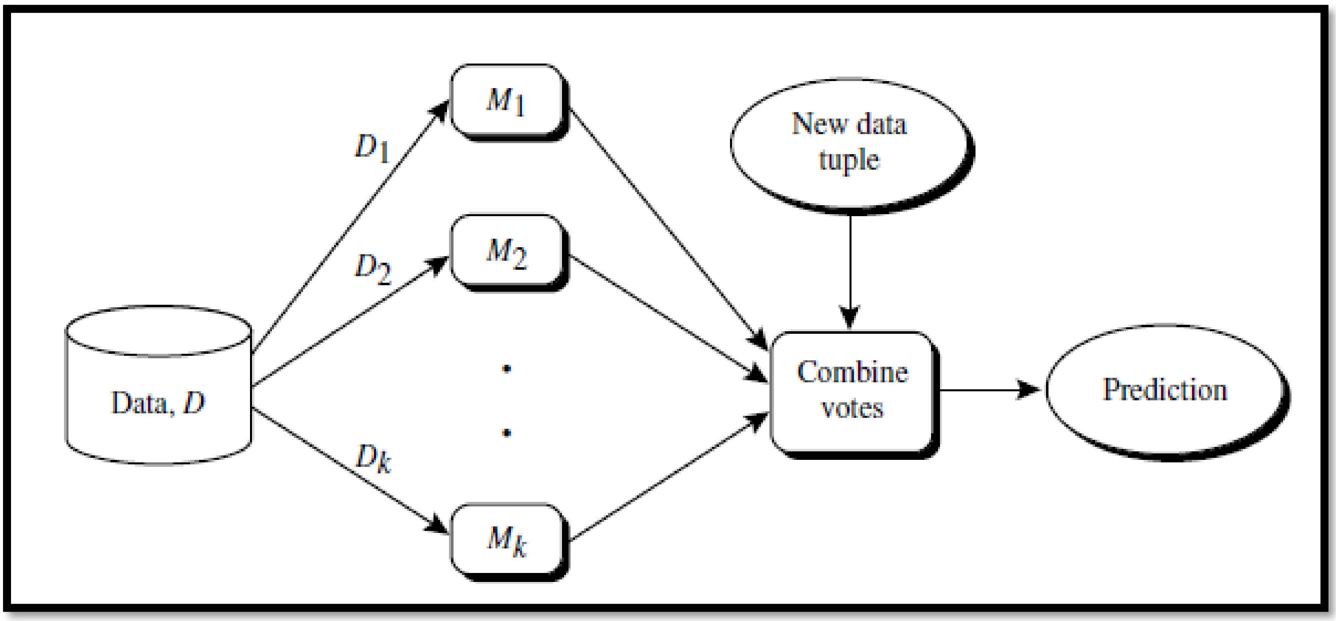


Techniques to Improve Classification Accuracy:

Ensemble Methods

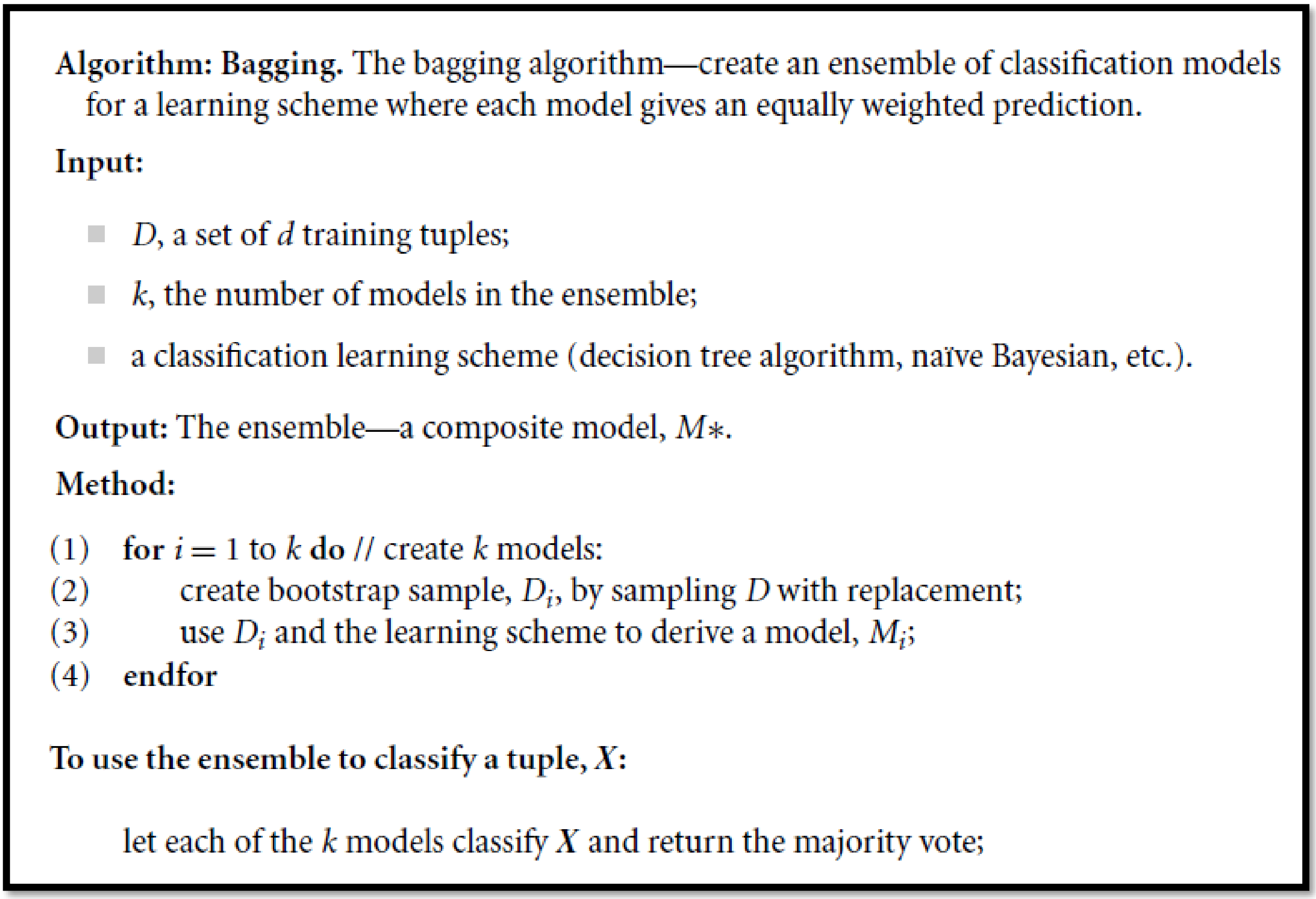
Bagging, boosting, and random forests are examples of ensemble methods.

An ensemble combines a series of k learned models (or base classifiers),M1, M2, … , Mk, with the aim of creating an improved composite classification model, M_. A given data set, D, is used to create k training sets, D1, D2, …, Dk, where Di is used to generate classifier Mi.
Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.



Bagging: Bootstrap Aggregation

- **Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group.**



Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

Method:

```
for $i = 1$ to $k$ do // create $k$ models:
    create bootstrap sample, $D_i$, by sampling $D$ with replacement;
    use $D_i$ and the learning scheme to derive a model, $M_i$;
endfor
```

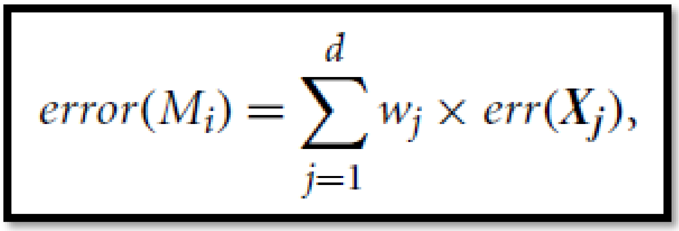
To use the ensemble to classify a tuple, X:

let each of the k models classify X and return the majority vote;

- **bagging stands for bootstrap aggregation. Each training set is a bootstrap sample, Because sampling with replacement is used, some of the original tuples of D may not be included in Di , whereas others may occur more than once.**

Boosting and AdaBoost

- Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses.
- In boosting, weights are also assigned to each training tuple. A series of k classifiers is iteratively learned. After a classifier, Mi , is learned, the weights are updated to allow the subsequent classifier,Mi+1, to "pay more attention" to the training tuples that were misclassified by Mi .
- The final boosted classifier, M*, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.
- To compute the error rate of model Mi , we sum the weights of each of the tuples in Di that Mi misclassified. That is,

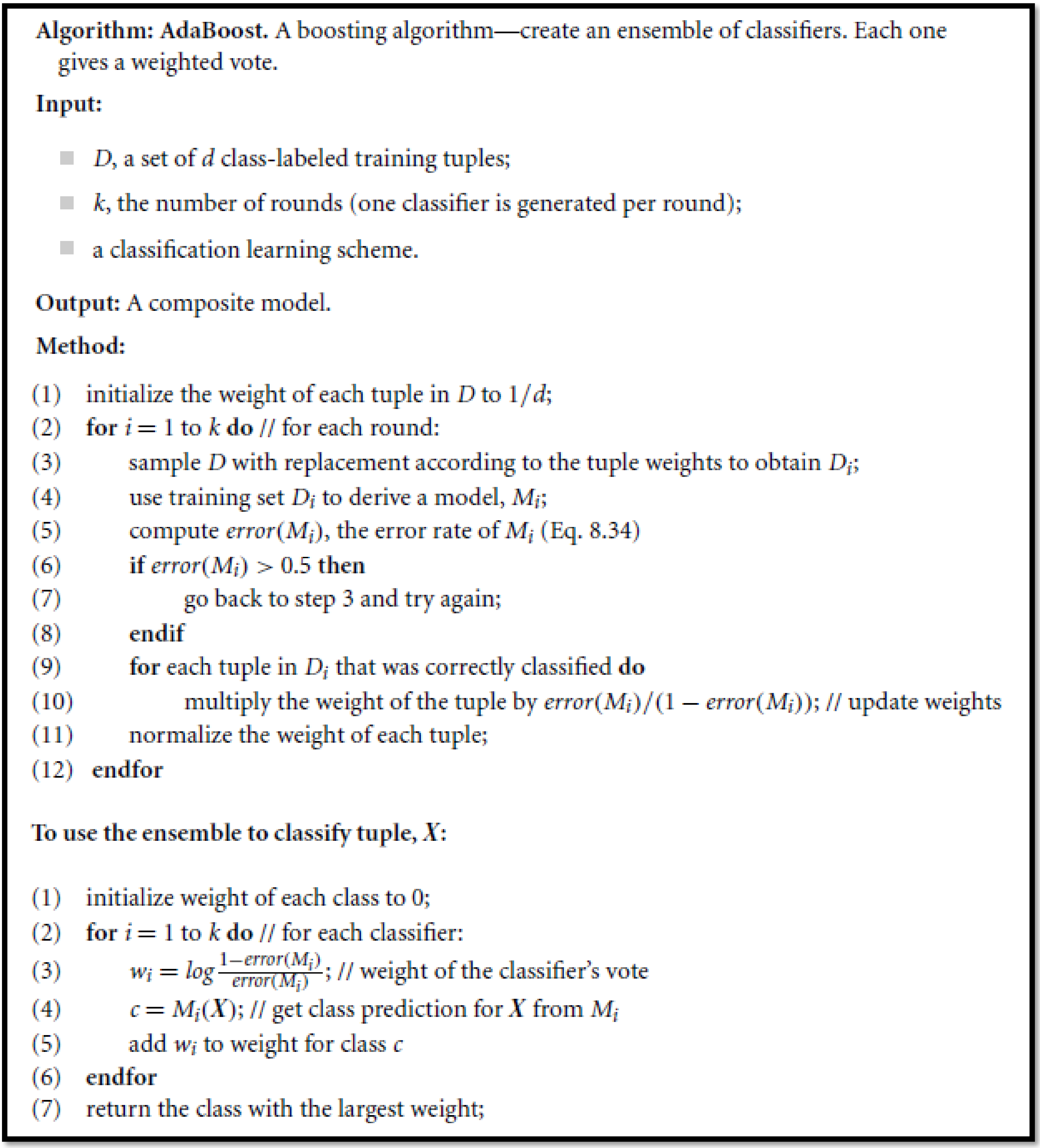


$$error(M_i) = \sum_{j=1}^{d} w_j \times err(X_j),$$

- err(Xj) is the misclassification error of tuple Xj: If the tuple was misclassified, then err(Xj) is 1; otherwise, it is 0.

Once boosting is complete, how is the ensemble of classifiers used to predict the class label of a tuple, X? Unlike bagging, where each classifier was assigned an equal vote, boosting assigns a weight to each classifier's vote, based on how well the classifier performed. The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be. The weight of classifier Mi 's vote is

$$\boxed{log \frac{1 - error(M_i)}{error(M_i)}.}$$



Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

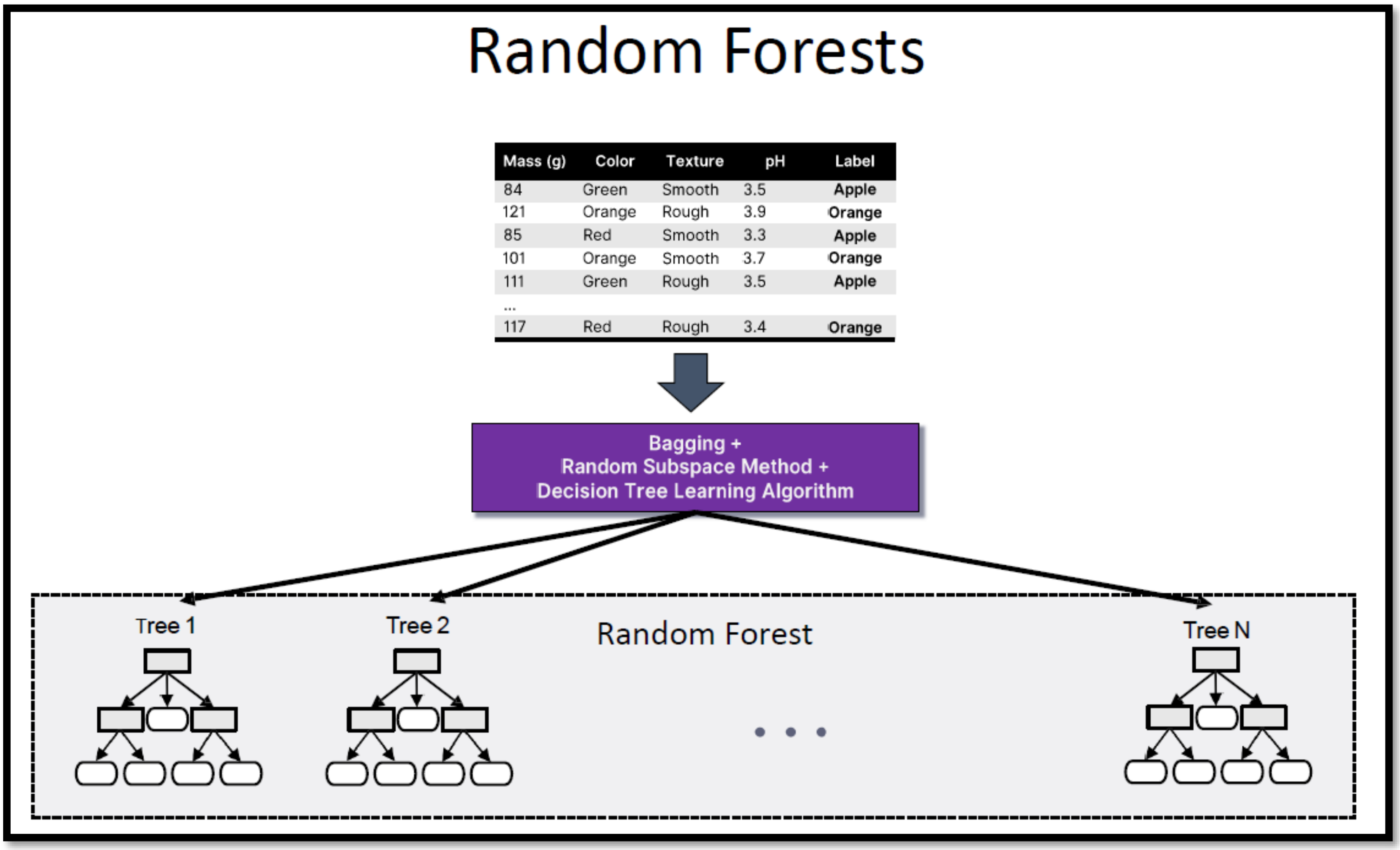
```
(1)   initialize the weight of each tuple in $D$ to $1/d$;
(2)   for $i = 1$ to $k$ do // for each round:
(3)       sample $D$ with replacement according to the tuple weights to obtain $D_i$;
(4)       use training set $D_i$ to derive a model, $M_i$;
(5)       compute $error(M_i)$, the error rate of $M_i$ (Eq. 8.34)
(6)       if $error(M_i) > 0.5$ then
(7)           go back to step 3 and try again;
(8)       endif
(9)       for each tuple in $D_i$ that was correctly classified do
(10)          multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11)      normalize the weight of each tuple;
(12)  endfor
```

To use the ensemble to classify tuple, X:

```
(1)   initialize weight of each class to 0;
(2)   for $i = 1$ to $k$ do // for each classifier:
(3)       $w_i = log \frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4)       $c = M_i(X)$; // get class prediction for $X$ from $M_i$
(5)       add $w_i$ to weight for class $c$
(6)   endfor
(7)   return the class with the largest weight;
```

Random Forests

- **Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a "forest." The individual decision trees are generated using a random selection of attributes at each node to determine the split During classification, each tree votes and the most popular class is returned.**
- A training set, D, of d tuples is given. The general procedure to generate k decision trees for the ensemble is as follows.
- For each iteration, i =(1, 2, …. , k), a training set, Di, of d tuples is sampled with replacement from D. That is, each Di is a bootstrap sample of D,
- so that some tuples may occur more than once in Di, while others may be excluded.
- Let F be the number of attributes to be used to determine the split at each node, where F is much smaller than the number of available attributes.



Random Forests

Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	Apple
121	Orange	Rough	3.9	Orange
85	Red	Smooth	3.3	Apple
101	Orange	Smooth	3.7	Orange
111	Green	Rough	3.5	Apple
...				
117	Red	Rough	3.4	Orange

Bagging +
Random Subspace Method +
Decision Tree Learning Algorithm

- To construct a decision tree classifier, Mi, randomly select, at each node, F attributes as candidates for the split at the node.

- The CART methodology is used to grow the trees. The trees are grown to maximum size and are not pruned. Random forests formed this way, with random input selection, are called Forest-RI.

- Random forests are comparable in accuracy to AdaBoost, yet are more robust to errors and outliers.

- The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem.

- The accuracy of a random forest depends on the strength of the individual classifiers and a measure of the dependence between them.

- The ideal is to maintain the strength of individual classifiers without increasing their correlation. Random forests are insensitive to the number of attributes selected for consideration at each split.