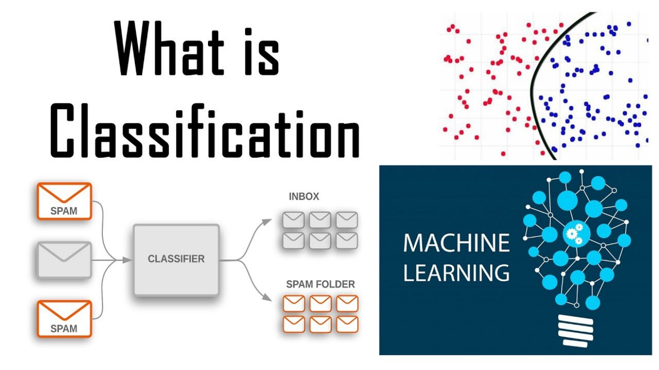
Table of contents

1. Introduction

Most of this material is derived from the text, Han, Kamber and Pei, Chapter 8, or the corresponding powerpoint slides made available by the publisher. Where a source other than the text or its slides was used for the material, attribution is given. Unless othewise stated, images are copyright of the publisher, Elsevier.

In this module of work we aim to introduce the data mining problems of *classification and prediction*, and to understand the basic classification technique of *decision trees*, so that you can recognise a problem to which they may apply; can apply them to the problem; and can evaluate the quality of the results.

ACTION: You can check out the first few minutes of this video for a brief description of what a classification problem looks like:



ACTION: You can watch this video lecture overview of the classsification and prediction topic if you find it helpful but all it covers is also in the written notes.

Prerecorded lecture for classification and prediction

2. Classification (Text: 8.1)

Classification builds models that describe interesting **classes** of data. The models are called **classifiers** because, once built, the model may be used to classify **unseen** data. Sometimes the model itself is more important than its use in ongoing classification because it provides a **compact summary** of the data, that is explanatory for humans.

Most commonly classification is **binary,** that is, objects are determined to belong to a class or not. For example, taxpayers are classified as fraudulent, or not. However, the generalisation to classfying data into more than two classes is important.

Classification is often classified as a machine-learning problem due to its origins in AI research, although data mining research has developed the scalability to handle large disk-stored data sets.

Nowadays it is widely used in application to problems in science, marketing, fraud detection, performance prediction, medical diagnosis, and fault diagnosis.

Classification

- Used to predict **categorical class labels** (discrete or nominal) from unlabelled data.
- Constructs (or **learns**) a **classifier** (or **model**) from **training** data that includes, for each **example** in the data, **data values** as well as a pre-determined **class label.**
- Uses the model to **predict** the class label for new,unseen, unlabelled data.

Classification vs Prediction

Although classifiers predict the values of unknown class labels, classification is usually distinguished from the problem of **numerical prediction** (commonly called simply *prediction*) that **builds models of continuous-valued functions** and so predicts unknown or missing **numeric** values. We will also study some popular prediction techniques.

Supervised Learning vs Unsupervised learning

Again, we see the AI influence in the language here, where **supervised** learning refers to classification as we have defined it -- where the training data (observations, measurements, etc.) is accompanied by labels indicating the class of the observations, and new data is classified based on the training set. In this AI-oriented view of classification we often talk about **batch** vs **incremental** learning. The former is usually an unstated assumption for data mining. In the latter case the labelled data becomes available to the learning algorithm in a sequence and a working classifier developed initially from a small amount of data must be continually updated to account for new data.

On the other hand, for **unsupervised** learning there are no class labels in the training data and the learning algorthm must find some interesting classes, or classifications with which to classify new data. This is commonly called **clustering.** We will study some popular clustering techniques later.

So classification can also be defined as **supervised learning of categorical variables.**

2.1. Two steps: Construct and Evaluate

Step 1: Training phase or learning step: Build a model from the labelled training set.

Each **tuple/sample/record/object/example/instance/feature vector** of the training dataset is assumed to belong to a predefined class, as determined by the class label attribute. Ideally, the tuples are a random sample from the full population of data.

- The set of tuples used for model construction is the *training set*:

  $T = \{X : X = (x_1, x_2, .., x_n)\}$ *and each* x_i *is an attribute value and* $X \in C_j$ *for some* $j = 1, .., k, k \geq 2$ *and* j *is the class label for* $X\}$.

- Commonly, each $X \in T$ is assumed to belong to exactly one class C_j
- In the very common special case of exactly 2 classes, i.e. binary learning, the training classes are called the **positive examples** C_+ or *P* and **negative examples** C_- or *N*.
- The model is represented as classification rules, decision trees, mathematical formulae, or a "black box". The model can be viewed as a function $f(X)$ that can predict the class label for some unlabelled tuple X.
- For classification models, the built model may be called a **classifier**.

Step 2: Use the model to classify unseen objects

- Need to estimate the **accuracy** of the model
  - The known labels of a set of independent **test samples** is compared with the classified results for those same samples from the model
  - **Accuracy** is the proportion of test set samples that are correctly classified by the model

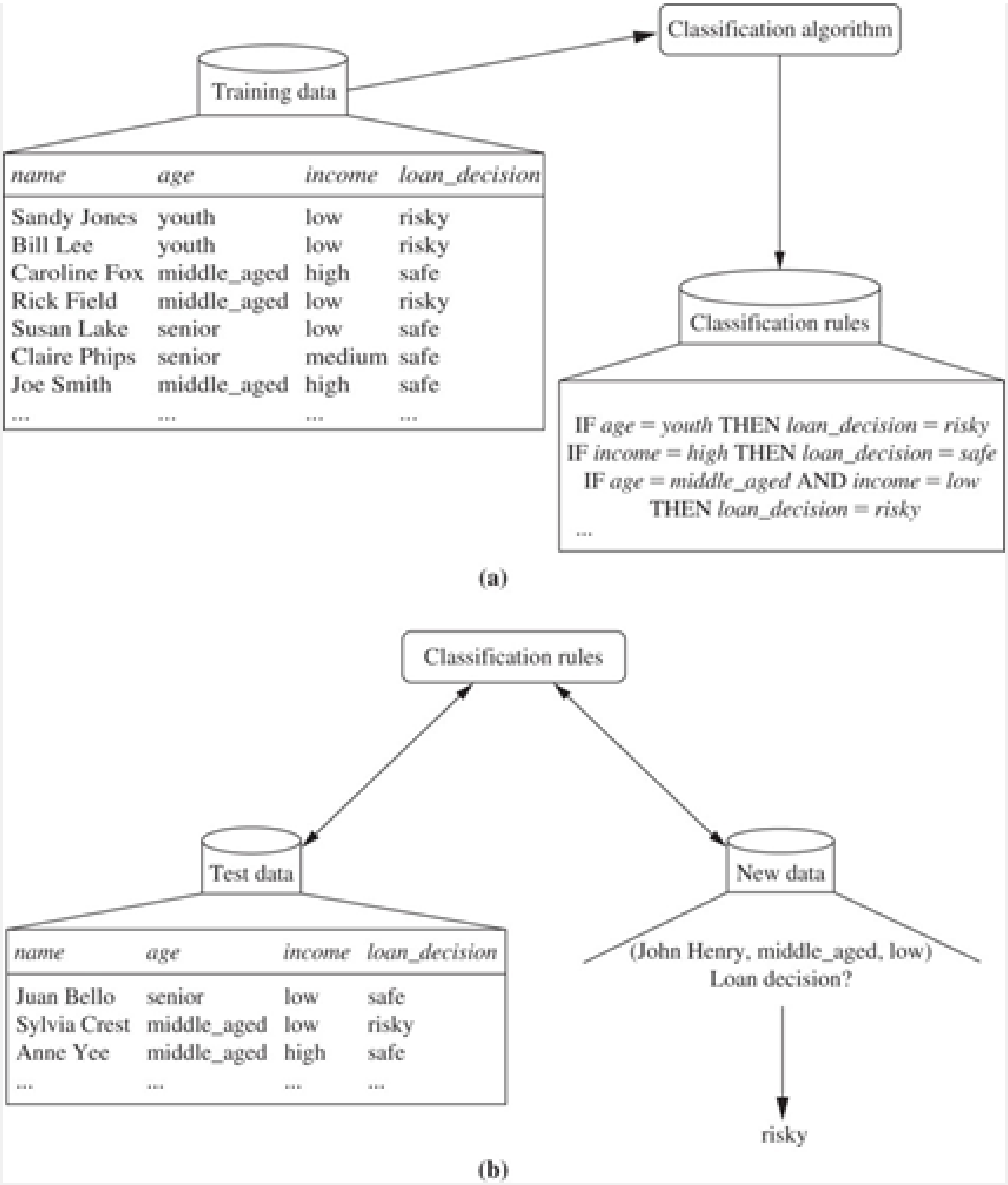
- If the accuracy and all other evaluation measures are acceptable, apply the model to classify data objects whose class labels are not known in the world.

Example:

The data classification process:

(a) Learning: Training data is analysed 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.

(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, unlabelled, data tuples.



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*
...

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

2.2. Evaluation

Learning algorithms (or *learners*) that build models built for classification and prediction are generally evaluated in the following ways. These ways are applied to the case of *inventing* new algorithms and wanting to assess them against others, or for *selecting* an algorithm for its suitability for a particular learning problem. Once an algorithm(s) has been selected and a model(s) built, these overarching principles may be revisited to choose which, if any, to put into practice.

- **Accuracy** often on *benchmark* data sets so they can be compared with other learning algorithms
  - Classifier accuracy: Predicting class label
  - Predictor accuracy: Guessing value of predicted attributes

- **Speed and complexity**
  - Time to construct the model (training time)
  - Time to use the model (classification/prediction time)
  - Worst case or average case theoretical complexity

- **Scalability**
  - Efficiency in handling disk-based databases
  - Potential for speed up by parallel computation

- **Robustness**
  - Handling noise and outliers

- **Interpretability**
  - Understanding and insight provided by the model

- **Other measures**
  - goodness of rules
  - decision tree size
  - compactness or simplicity of model

3. Decision Tree Induction (Text: 8.2)

A decision tree classifies labelled data by applying a sequence of logical tests on attributes that partition the data into finer and finer sets. The model that is learnt is a tree of logical tests.

The decision tree is a **flowchart-like structure**, where

- each **internal node** as well as the topmost **root node** represents a test on an attribute; commonly the tests can have only two outcomes, in which case the tree is **binary**
- each **branch** directed out and down from an internal node represents an outcome of the test
- each **leaf node** (or terminal node) represents represents a decision and holds a class label
- a **path** from the root to a leaf traces out the classification for a tuple

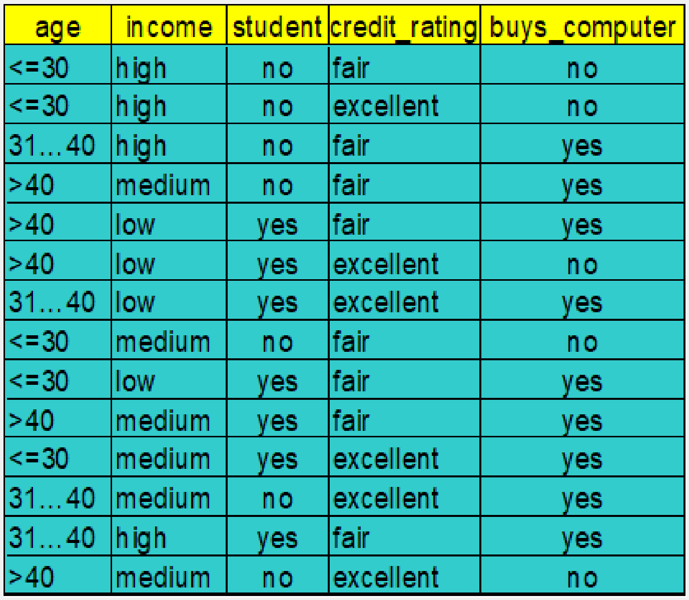
Decision tree induction is **very popular for classification** because:

- relatively fast learning speed
- convertible to simple and easy to understand classification rules
- can work with SQL queries to access databases while tree-building
- comparable classification accuracy with other methods

Exercise

Here is some data for a binary classification problem, with label *buys_computer*.

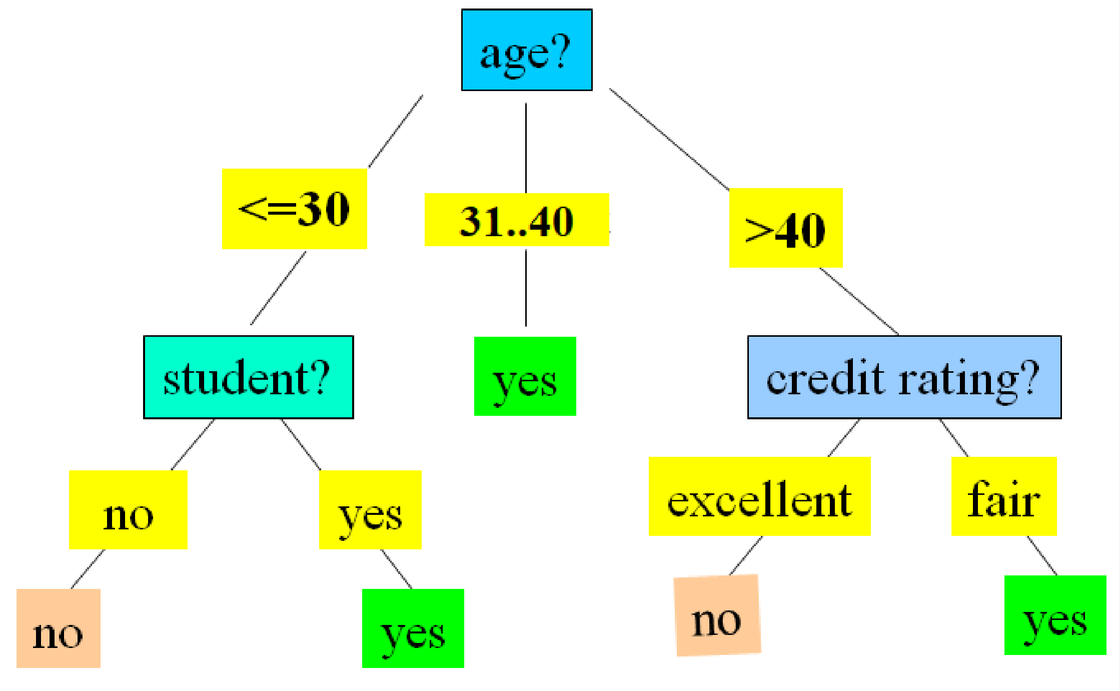
ACTION: Consider what sequence of decisions you would propose to identify "who buys a computer"? Is your tree binary, or not?



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

And here is a decision tree model to classify the data.

ACTION: Consider, how does it differ from yours? Can you always design a tree that correctly classifies every object in the training dataset?


age?
<=30
31..40
>40
student?
yes
credit rating?
no
yes
excellent
fair
no
yes
no
yes

4. Basic, greedy, decision tree algorithm

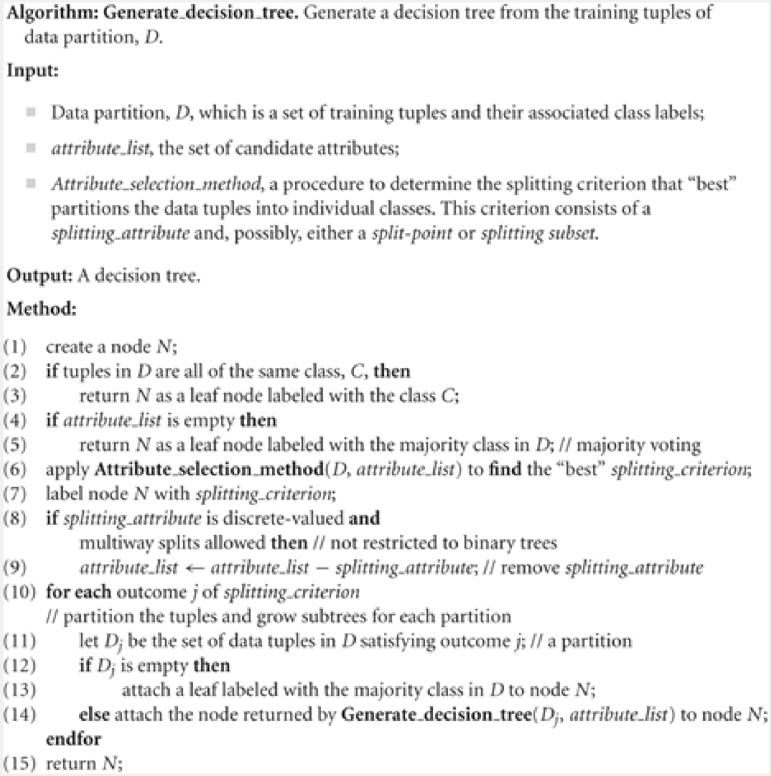
A typical basic algorithm follows. It is **greedy** (it makes decisions optimising the next step context and never backtracks to reconsider).

It is a **recursive**, top-down divide-and-conquer approach to build a tree.

Attributes may be **nominal, ordinal,** or **continuous.**

- At the **start,** all the training examples are at the root
- At a node, **test attributes are selected** on the basis of a heuristic or statistical measure (e.g., information gain)
- Examples at the node are **partitioned** to sub-nodes based on selected attributes
- **Recurse** over subnodes
- Paritioning **stops** when
  - All samples for a given node belong to the same class; or
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf; or
  - There are no samples left

The slightly more generic algorithm sketch below permits n-ary (*multiway*) trees and can discretise continuous attributes dynamically, according to local context in the tree.



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;
```

ACTION: Go back to the previous page and build a tree stepping through the algorithm as shown here.

Is your tree any different this time? What attribute_selection_method did you use?

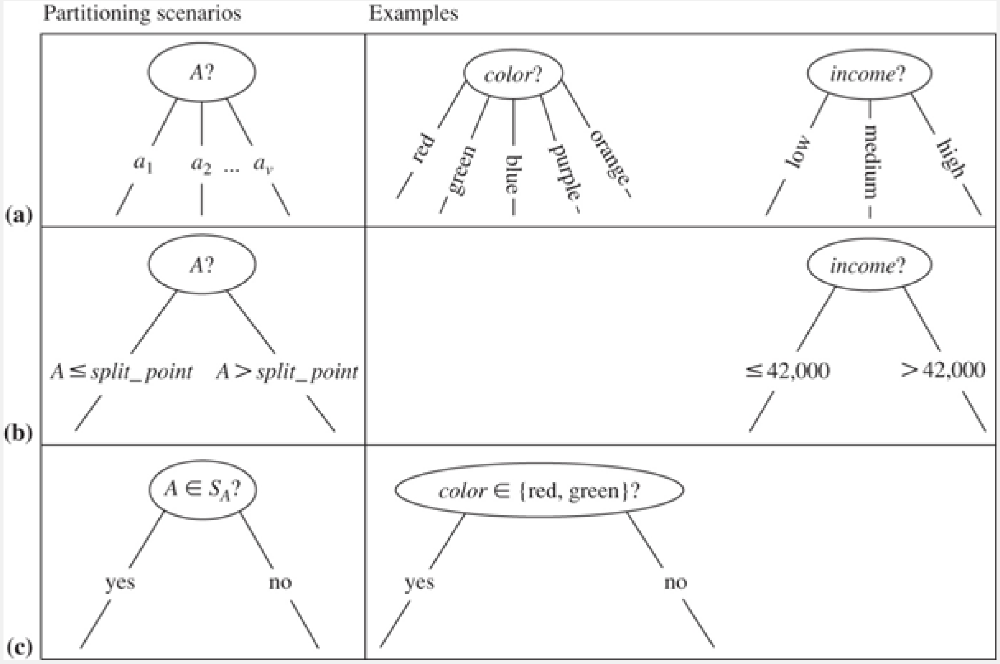
5. Attribute Selection Methods (Text 8.2.2)

Attribute selection methods are also called **splitting rules.**

- Techniques to choose a **splitting criterion** comprised of a **splitting attribute** and a **split point** or **splitting subset**
- Aim to have partitions at each branch as **pure** as possible -- i.e. all examples at each sub- node belong in the same class.

Example

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 nominal-valued, then one branch is grown for each known value of A. (b) If A is continuous-valued or ordinal, then two branches are grown, corresponding to A $\leq$split_point and A > split_point. (c) If A is nominal and a binary tree must be produced, then the test is of the form A $\in S_A$, where S_A is the splitting subset for A.



Heuristics, (or **attribute selection measures**) are used to choose the best splitting criterion.

- Information Gain, Gain ratio and Gini index are most popular.
- Information gain:
  - biased towards multivalued attributes
- Gain ratio:
  - tends to prefer unbalanced splits in which one partition is much smaller than the others
- Gini index:
  - biased towards multivalued attributes
  - has difficulty when number of classes is large
  - tends to favour tests that result in equal-sized partitions and purity in both partitions

5.1. Information Gain

Information Gain

- This was a very early method that sprang from AI research in ID3, and was refined further to become *Gain Ratio* in C4.5.
- It selects the attribute to split on with the highest information gain

- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, of m classes, where $C_{i,D}$ is the set of tuples in D labelled with class C_i
    - estimated by $p_i = \frac{|C_{i,D}|}{|D|}$
- Expected information (**entropy**) needed to classify a tuple in $D$is defined by

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

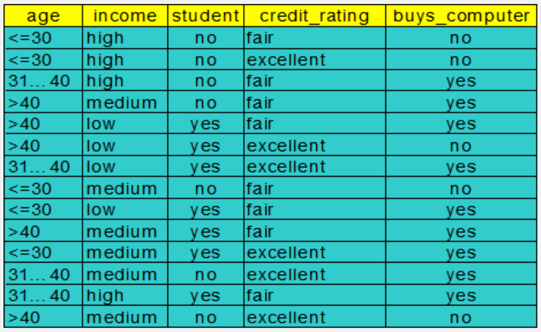
- After using attribute A to split D into v partitions, corresponding to each attribute value for A, each one of these partitions being D_j, the information that is still needed to separate the classes is:

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

- Therefore, information gained by branching on attribute A is given by

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

Example (continued from previous)



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

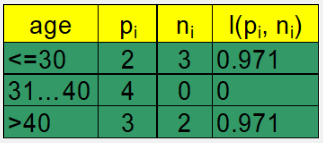
Consider 2 classes: Class *P* is buys_computer = "yes". Class *N* is buys_computer = "no"

For some partition on D with p_i examples of *P* and n_i examples of *N*, let $Info(D)$ be written as $I(p_i, n_i)$.

Using the definition $Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i)$ from above,

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

Now consider the first partition on *age*. We have the following



age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

and so

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

= 0.694

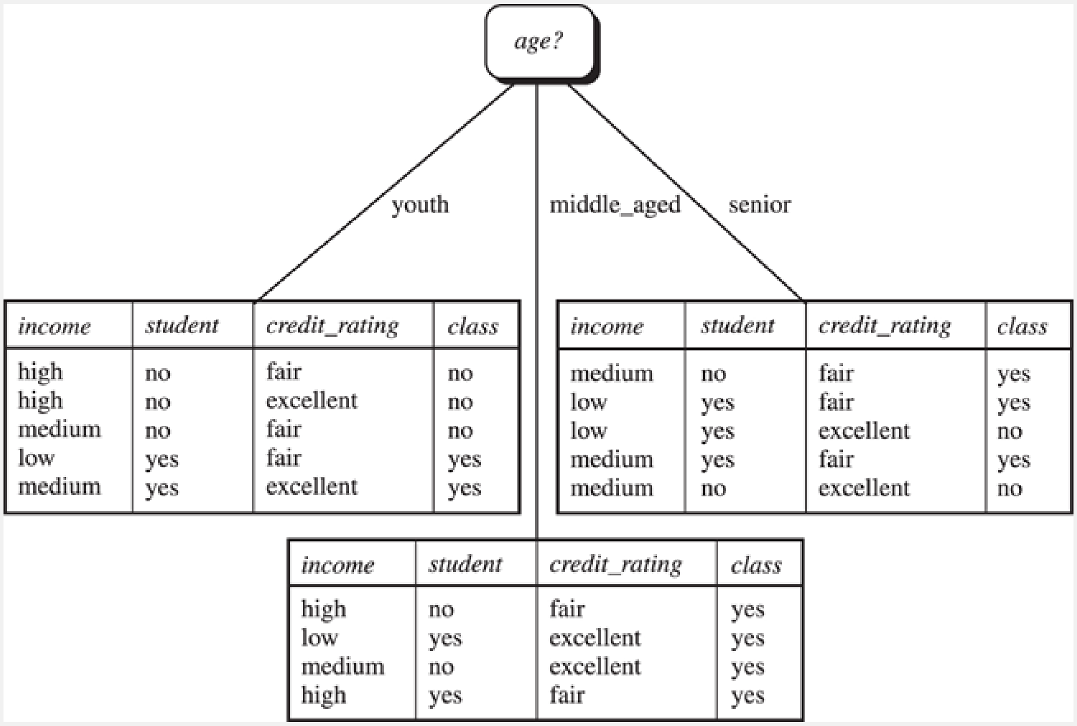
Therefore

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

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

So Gain(age) is optimal and we split on *age* to get



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

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

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

5.2. Information Gain for continuous-valued attributes

Let attribute *A* be a continuous-valued attribute
To determine the *best split point* for A

- Sort the values of *A* in increasing order
- Typically, the **midpoint between each pair of adjacent values** is considered as a possible split point
  - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
- Of these, the point with the minimum expected information requirement for *A*, $Info_A(D)$ is selected as the split-point for *A*

Then Split:

D1 is the set of tuples in *D* satisfying *A* ≤ *split-point*, and *D2* is the set of tuples in *D* satisfying *A* > *split-point*

This method can also be used for ordinal attributes with many values (where treating them simply as nominals may cause too much branching).

5.3. Gain Ratio

- Used in C4.5 (a successor of ID3) to overcome bias towards attributes with many values
- Normalises information gain

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

splitInfo$_A$ represents the *potential* information generated by splitting *D* into *v* partitions, corresponding to the *v* outcomes of a test on *A*.

Now we define

> *GainRatio(A) = Gain(A)/SplitInfo(A)*

<u>Example</u> (continued from previous):

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

> *gain_ratio(income) = 0.029/1.557 = 0.019*

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

5.4. Gini index

Gini index is used in CART and IBM Intelligent miner decision tree learners

- All attributes are assumed **nominal**

If a data set D contains examples from n classes, gini index, $gini(D)$, measures the **impurity** of D and is defined as

$gini(D) = 1 - \sum_{j=1}^{n} {p_j}^2$ where p_j is the relative frequency of class j in D

If a data set D is split on attribute A into two subsets D_1 and D_2, the gini index $gini_A(D)$ is defined as the size-weighted sum of the impurity of each partition:

$$gini_A(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2)$$

To split a node in the tree:

- Enumerate all the possible ways of splitting all the possible attributes
- The attribute *split* that provides the smallest *gini$_{split}$(D)* (i.e the greatest purity) is chosen to split the node

Example (continued from previous)
D has 9 tuples in class buys_computer = "yes" and 5 in "no"

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

Now consider the attribute *income*. Partition *D* into 10 objects in *D1* with income in *{low, medium}* and 4 objects in *D2* with income in *{high}*

We have

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

Similarly, *gini$_{income}${low,high}* is 0.458; and gini$_{income}${medium,high} is 0.450.

Thus, we split on the {low,medium} (and the other partition is {high}) since it has the lowest Gini index

When attributes are **continuous or ordinal**, the method for selecting the **midpoint between each pair of adjacent values** (described earlier) may be used.

5.5. Other Attribute Selection Methods

Of course, researchers have experimented with many other attribute selection methods that you might come across. Here are some of the most well-known that you might like to look into further.

- **CHAID**: a popular decision tree algorithm, measure based on χ2 test for independence
- **C-SEP**: performs better than info. gain and gini index in certain cases
- **G-statistic**: has a close approximation to χ2 distribution
- **MDL (Minimal Description Length) principle** (i.e., the simplest solution is preferred):
    - The best tree as the one that requires the fewest number of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations), e.g. **CART**: finds multivariate splits based on a linear combination of attributes.

6. Overfitting and tree pruning (Text 8.2.3)

If you have consistently labelled data, and you allow attributes to be re-used when growing the tree, and you stop growing when every training tuple is classified, it is always possible to have a tree that describes the training set with 100% accuracy. Such a tree typically has very poor *generalisation* and is said to *overfit* the training data.

Overfitting: An induced tree may **overfit** the training data

- Too many branches, some may reflect anomalies due to noise or outliers
- Poor accuracy for unseen samples is observed because anomalies are modelled and objects *like* the training data are not modelled. 100% accuracy on the training data can be a *bad thing* for accuracy on unseen data.

There are two typical approaches to avoid overfitting:

- **Prepruning**: Stop tree construction early - do not split a node if this would result in the goodness measure falling below a threshold. But it is difficult to choose an appropriate threshold.
- **Postpruning**: Remove branches from a "fully grown" tree. This produces a sequence of progressively pruned trees. Then use a set of data different from the training data to decide which is the "best pruned tree"

> **ACTION:** Overfitting is an important but rather qualitative, loosely-defined concept. If you need more explanation of overfitting, refer to https://en.wikipedia.org/wiki/Overfitting

7. Enhancements to the Basic Algorithm (not in text)

These enhancements may be embedded within the decision tree induction algorithm

- Handle missing attribute values
    - Assign the most common value of the attribute
    - Assign probability to each of the possible values
- Attribute construction
    - Create new attributes based on existing ones that are sparsely represented
    - This reduces fragmentation, repetition, and replication
- Continuous target variable
    - In this case the tree is called a *regression tree*, the leaf node classes are represented by their mean values, and the tree performs prediction (using that mean value) rather than classification.
- Probabilistic classifier
    - Instead of majority voting to assign a class label to a leaf node, the *proportion* of training data objects of each class in the leaf node can be interpreted to as the *probability* of the class, and this probability can be assigned to the classification for unknown objects falling in that node at use-time.

ACTION: There is a nice non-technical overview of decision trees, that you might like to read if you are needing more:

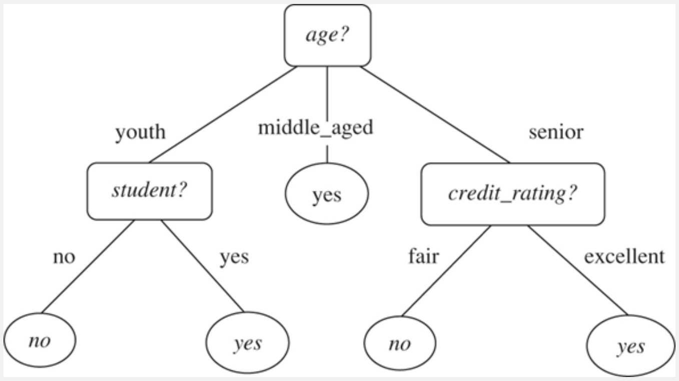
https://algobeans.com/2016/07/27/decision-trees-tutorial/

8. Extracting Rules from Decision Trees (Text 8.4.2)

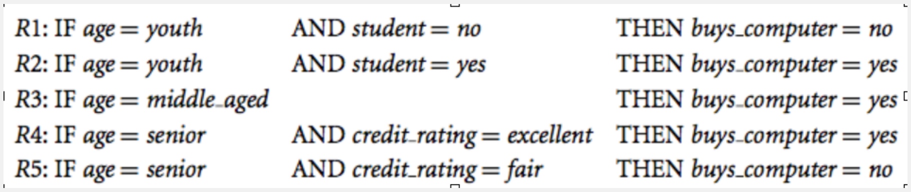
Decision trees can become large and difficult to interpret. We look at how to build a rule-based classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction

<u>Example:</u>



From the decision tree above, we can extract IF-THEN rules by tracing them from the root node to each leaf node:



R1: IF *age* = *youth*	AND *student* = *no*	THEN *buys_computer* = *no*
R2: IF *age* = *youth*	AND *student* = *yes*	THEN *buys_computer* = *yes*
R3: IF *age* = *middle_aged*		THEN *buys_computer* = *yes*
R4: IF *age* = *senior*	AND *credit_rating* = *excellent*	THEN *buys_computer* = *yes*
R5: IF *age* = *senior*	AND *credit_rating* = *fair*	THEN *buys_computer* = *no*

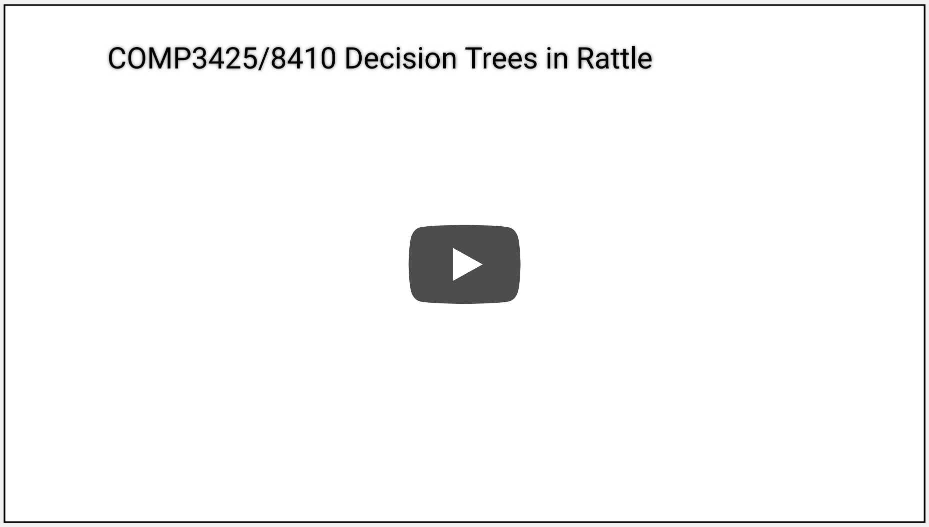
Properties of extracted rules

- Mutually exclusive
  - We cannot have rule conflicts here because no two rules will be triggered for the same tuple.
  - We have one rule per leaf, and any tuple can map to only one leaf.
- Exhaustive:
  - There is one rule for each possible attribute–value combination, so that this set of rules does not require a default rule.
  - The order if rules is irrelevant.

* Rattle can generate rules from a trained decision tree.

9. Practical Exercise: Decision trees

ACTION: Do this practical exercise. A video showing the use of the Rattle functions needed is provided to assist you.



COMP3425/8410 Decision Trees in Rattle

Practical Exercise: Decision trees