Lecture 4
Classification

CA4010: Data Warehousing and Data Mining
2016/2017 Semester 1

Dr. Mark Roantree
Dublin City University

Agenda

1. **Overview**
2. **Bayes Classifiers**
3. **Nearest Neighbour**
   - Distance Measures
   - Normalisation
4. **Decision Trees**
   - Top Down Induction of Decision Trees
   - Attribute Selection
   - Alternate Decision Trees
5. **Attribute Selection Measures**
   - Information Gain
   - Gain Ratio
   - Gini Index
     - Gini Index Example

What is Classification?

- A bank needs analysis of data in order to learn which loan applicants are *safe* and which are *risky* for the bank.
- A marketing manager needs data analysis to help guess whether a customer with a given profile will buy a new computer.
- A medical researcher wants to analyze breast cancer data in order 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 categorical labels, such as *safe* or *risky* for the loan application data; *yes* or *no* for the marketing data; or treatment *A*, *B*, or *C* for the medical data.
- These categories can be represented by discrete values, where the ordering among values has no meaning.

What is Prediction?

- Suppose that the marketing manager would like to predict how much a given customer will spend during a sale.
- This data analysis task is an example of numeric prediction, where the model constructed predicts a *continuous-valued* function, or *ordered value*, as opposed to a *categorical label*.
- This model is a **predictor**.
- In each of these examples, the data analysis task is **classification**, where a *model* or *classifier* is constructed to predict categorical labels, such as *safe* or *risky* for the loan application data; *yes* or *no* for the marketing data; or treatment *A*, *B*, or *C* for the medical data.
- These categories can be represented by discrete values, where the ordering among values has no meaning.

How does classification work?

- Data classification is a two-step process: figs 1,2.
- In the first step, a **classifier** is built describing a predetermined set of data classes or concepts.
- This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or *learning from* a training set made up of database tuples and their associated class labels.
- A tuple *X*, is represented by an *n*-dimensional attribute vector X = (x_1, x_2,..,x_n), depicting *n* measurements made on the tuple from *n* database attributes, respectively A_1, A_2,..,A_n.
- Each tuple *X*, is assumed to belong to a predefined class as determined by another database attribute called the **class label** attribute.
- The class label attribute is *discrete-valued* and *unordered*.
- It is *categorical* in that each value serves as a category or class.

Data Classification Process

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.

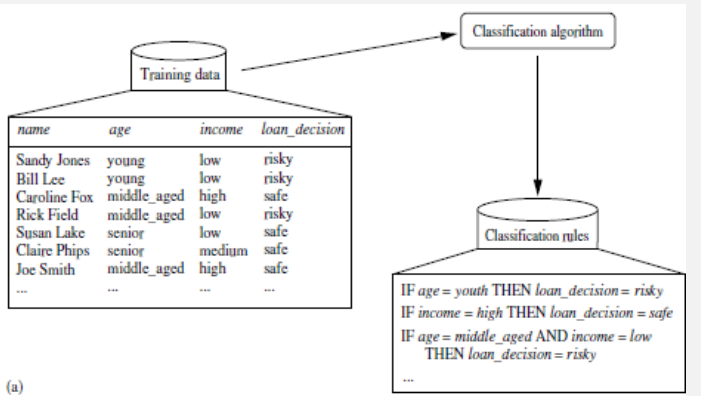


Figure 1: Data Classification: Learning

How does classification work?

- The tuples in the training set are referred to as **training tuples** and are selected from the database under analysis.
- In the context of classification, data tuples are samples, examples, instances, data points, or objects.
- Because the **class label** of each training tuple is provided, this step is also known as **supervised learning**: 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 or set of classes to be learned may not be known in advance.
- 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.

Data Classification Process

Classification: Test data are used to estimate the *accuracy* of the classification rules. If acceptable, the rules are used to classify new data.

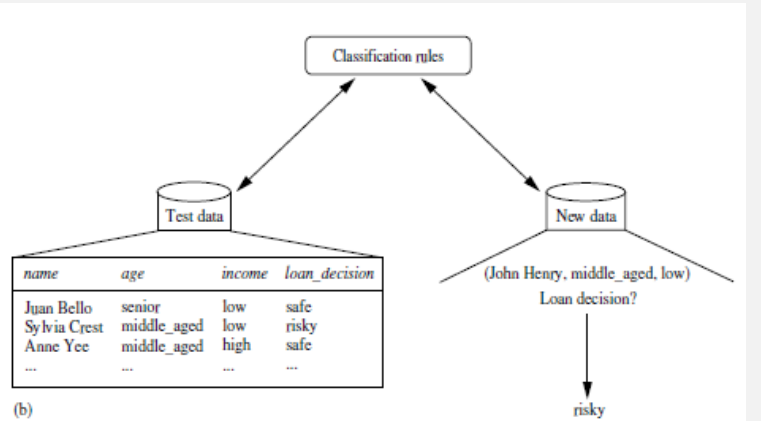


Figure 2: Data Classification

What about classification accuracy?

- In the second step (Figure 2), the model is used for *classification*.
- First, the *predictive accuracy* of the classifier is estimated.
- If we were to use the training set to measure the accuracy of the classifier, this estimate would likely be **optimistic**.
- This is because the classifier tends to *overfit the data*: 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.
- These tuples are randomly selected from the general data set.
- They are independent of the training tuples, meaning that they are not used to construct the classifier.

Accuracy & Unseen Data

- The **accuracy** of a classifier on a given test set is the *percentage* of test set tuples that are *correctly classified*.
- The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple.
- If this accuracy is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known.
- We refer to this as *unknown* or previously *unseen* data.
- For example, the classification rules learned in Figure 1 from the analysis of data from previous loan applications can be used to approve or reject new or future loan applicants.

Numeric Prediction

- Data prediction is a two-step process, similar to that of data classification.
- With prediction, we lose the terminology of **class label attribute** because the attribute for which values are being predicted is **continuous-valued** (ordered) rather than categorical (discrete-valued and unordered).
- The attribute is simply the *predicted attribute*.
- Assume we wish to predict the amount that would be "safe" for the bank to loan an applicant.
- The data mining task becomes **prediction**, rather than classification.
- We would replace the categorical attribute "loan decision", with the continuous-valued "loan amount" as the predicted attribute, and build a *predictor*.

Different Models

- Prediction and classification also differ in the methods that are used to build their respective models.
- As with classification, the training set used to build a predictor should not be used to assess its accuracy.
- An independent test set is required.
- The accuracy of a predictor is estimated by computing an *error* based on the *difference* between the **predicted value** and the **actual known value** of y for each of the test tuples X.

Bayes Classifiers

- Classification is the process of **learning a model** that describes different classes of data.
- The classes are **predetermined**.
- For example, in a banking application, customers who apply for a credit card may be classified as a *poor risk*, *fair risk*, or *good risk*.
- Hence this type of activity is also called *supervised learning*.

Data Models

- Once the model is built, it can be used to classify new data.
- The first step (learning the model) is accomplished by using a training set of data that has already been classified.
- Each record in the training data contains an attribute, called the *class label*, to indicate the class to which the record belongs.
- The model that is produced is usually in the form of a *decision tree* or a *set of rules*.

Building the Model: Issues

- Some of the important issues with regard to the model and the algorithm that produces the model include:
  - the model's ability to predict the correct class of new data;
  - the computational cost associated with the algorithm;
  - and the scalability of the algorithm.

Everyday Classification

- Divide a group of objects so that each one is assigned to one of a number of mutually *exhaustive* and *exclusive* categories known as classes.
- Mutually exhaustive and exclusive = assigned to one (only) class.
- Players who play for teams in the same competition.
- Items in the fridge or in the press
- In DCU? Students in CASE4 or SS4.

The Basics

- Utilises probability theory to determine the most likely classification
- Using the 7am Dublin-Cork train, the probability it arrives *on-time* is between 0 and 1,
  where 0 is *impossible* and 1 is *certain*.
- A probability of 0.7 means that after a long series of observations (N tests), the probability the train arrives on time is 0.7.
- The greater the value of N, the higher your confidence or reliability of your probability.

Multiple Events

A single event does not provide true classification

- Consider again our train example.
- Typically, one would define 4 *mutually exclusive and exhaustive* events.

Multiple events and their Probabilities P(e)

E1	train cancelled	P(E1)	=0.05
E2	train 10mins or more + late	P(E2)	=0.1
E3	train less than 10mins late	P(E3)	=0.15
E4	train on time	P(E4)	=0.7

Conditions

- Each of the conditions *E1* to *E4* must be between 0 and 1
- The sum of all conditions must equal 1.
- $P(\text{E1}) + P(\text{E2}) + P(\text{E3}) + P(\text{E4}) = 1$
- In general, the **sum of probabilities** of a set of *mutually exclusive and exhaustive* events must be 1.

Training Sets

- In reality, we cannot measure every single event and instead use a sample (eg. 100 days).
- The terminology used for sample datasets is a *training set*.
- A **Training Set** constitutes the results from sample trials that we use to *predict the classification* of other (unclassified) instances.
- In figure 3, the training set has 20 instances, with **4 attributes** and a **classification**.

Training Set Example

day	season	wind	rain	class
weekday	spring	none	none	on time
weekday	winter	none	slight	on time
weekday	winter	none	slight	on time
weekday	winter	high	heavy	late
saturday	summer	normal	none	on time
weekday	autumn	normal	none	very late
holiday	summer	high	slight	on time
sunday	summer	normal	none	on time
weekday	winter	high	heavy	very late
weekday	summer	none	slight	on time
saturday	spring	high	heavy	cancelled
weekday	summer	high	slight	on time
saturday	winter	normal	none	late
weekday	summer	high	none	on time
weekday	winter	normal	heavy	very late
saturday	autumn	high	slight	on time
weekday	autumn	none	heavy	on time
holiday	spring	normal	slight	on time
weekday	spring	normal	none	on time
weekday	spring	normal	slight	on time

Figure 3: The Train dataset

Training Set Probabilities

Probabilities for all Events
Lets do some calculations . . . 20 instances in our dataset

Event	Description	Total	P(E)
E1	cancelled	1	0.05
E2	very late	3	0.15
E3	late	2	0.1
E4	on time	14	0.7

Method 1: Simple

Searching for: {weekday,winter,high,heavy}?

- Look at the frequency of each classification and pick the most common: *on time*
- The Flaw?
- All *unseen* instances will be classified in identical fashion
- Not bad, we will be correct 70% (0.7) of the time!
- But the Goal: make correct predictions as often as possible!

Using Probability: Prior Probability

- This training set contains the classification (E) in addition to *4 other attributes*
- Lets assume they are recorded for a reason: they affect the outcome!
- **Definition**. The probability of the train being on time, calculated using the frequency of *on time* divided by the total instances is known as the **prior probability**.
- With no other info: P(class = on time) = 14/20 = 0.7
- To effectively use additional attributes, we must introduce *conditional probability*!

Using Probability: Conditional Probability

- What is the probability of the train being *on time* if the season is *Winter*?
- In other words: probability that they occur *in the same instance*?
- Calculation: the number of co-occurrences of *on time* and *Winter* divided by the number of occurrences of *Winter*, ie. 2/6 = 0.33.
- 0.33 is a lot less than 0.7 but it seems intuitive as trains run late, more often, in Winter!

Conditional Probability

- **Definition**. The probability of an event occurring if we know that an attribute has a particular value (or several variables have particular values) is called the **conditional probability** of the event occurring.
- $P(\text{class} = \text{on time} \mid \text{season} = \text{winter})$
- The probability that the class is *on time* **given that** the season is *winter*.

Using Probability: Posterior Probability

- P(class = on time | season = winter) is also called a **posterior probability**.
- It means that one can calculate for the classification *after* obtaining the information that the season is *Winter*.
- By contrast, the **prior probability** is that estimated *before* any information is available.

The Unseen Instance

P(class = on time | day = weekday and season = winter and wind = high and rain = heavy)

- To calculate the most likely classification for the *unseen* instance given previously we could calculate the probability using the above equation.
- Then also calculate *P* for the other classifications (late, very late, cancelled)
- However, there are only 2 instances with these combinations and thus, unreliable.

Method 2: A More Reliable Estimate

Use conditional probabilities based on a single attribute

$P(\text{class} = \text{on time} \mid \text{season} = \text{winter}) = 2/6 = 0.33$
$P(\text{class} = \text{late} \mid \text{season} = \text{winter}) = 1/6 = 0.17$
$P(\text{class} = \text{very late} \mid \text{season} = \text{winter}) = 3/6 = 0.5$
$P(\text{class} = \text{cancelled} \mid \text{season} = \text{winter}) = 0/6 = 0.0$

- As the third P has the largest value, we could conclude that the most likely classification is *very late*.
- This is a different result to our first approach!
- We could do similar calculations with attributes *day*, *rain* and *wind*.
- This may provide a different result again! Which is best?

Naive Bayes Classifiers

A Means of combining prior probability and conditional probabilities into a single formula

- Strategy: Use this method for each classification in turn and choose the classification with the largest value.
- We invert our previous formula: use the conditional probability that the season is *winter* given the class is *very late*.
- P(season = winter | class = very late)

calculated as the **count** of season=*winter* and class=*very late* in the **same instance**,
divided by the **number of instances** for which the class is *very late*

Conditional and Prior Probabilities

	class = on time	class = late	class = very late	class = cancelled
day = weekday	**9/14 = 0.64**	**1/2 = 0.5**	**3/3 = 1**	**0/1 = 0**
day = saturday	2/14 = 0.14	1/2 = 0.5	0/3 = 0	1/1 = 1
day = sunday	1/14 = 0.07	0/2 = 0	0/3 = 0	0/1 = 0
day = holiday	2/14 = 0.14	0/2 = 0	0/3 = 0	0/1 = 0
season = spring	4/14 = 0.29	0/2 = 0	0/3 = 0	1/1 = 1
season = summer	6/14 = 0.43	0/2 = 0	0/3 = 0	0/1 = 0
season = autumn	2/14 = 0.14	0/2 = 0	1/3 = 0.33	0/1 = 0
season = winter	**2/14 = 0.14**	**2/2 = 1**	**2/3 = 0.67**	**0/1 = 0**
wind = none	5/14 = 0.36	0/2 = 0	0/3 = 0	0/1 = 0
wind = high	**4/14 = 0.29**	**1/2 = 0.5**	**1/3 = 0.33**	**1/1 = 1**
wind = normal	5/14 = 0.36	1/2 = 0.5	2/3 = 0.67	0/1 = 0
rain = none	5/14 = 0.36	1/2 =0.5	1/3 = 0.33	0/1 = 0
rain = slight	8/14 = 0.57	0/2 = 0	0/3 = 0	0/1 = 0
rain = heavy	**1/14 = 0.07**	**1/2 = 0.5**	**2/3 = 0.67**	**1/1 = 1**
Prior Probability	**14/20 = 0.70**	**2/20 = 0.10**	**3/20 = 0.15**	**1/20 = 0.05**

Figure 4: Conditional and Prior Probabilities: train dataset

Posterior Probability

We have now calculated all conditional and prior probabilities in figure 5

- Prior Probabilities
  - class = on time: 14/20 = 0.70
  - class = late: 2/20 = 0.10
  - class = very late: 3/20 = 0.15
  - class = cancelled: 1/20 = 0.05
- Posterior Probabilities
  - These are the probabilities of real interest
  - The posterior probabilities of *each possible class* occuring for *a specified instance* i.e. for known values of *all* attributes

Calculating Posterior Probabilities

- Given a set of k mutually exclusive and exhaustive classifications c_1,c_2,... c_k, which have prior probabilities $P(c_1)$, $P(c_2)$,... $P(c_k)$, respectively;
- n attributes a_1,a_2,... a_k, which for a given instance have values v_1,v_2,... v_k, respectively, the posterior probability of class c_i occurring for the specified instance can be shown to be proportional to:

$$P(c_i) \times P(a_1 = v_1 \wedge a_2 = v_2 \cdots \wedge a_n = v_n \mid c_i) \quad (1)$$

Naïve Bayes Classification

If we assume the attributes to be independent, the value of this expression can be calculated using the product:

$$P(c_i) \times P(a_1 = v_1 \mid c_i) \times P(a_2 = v_2 \mid c_i) \times \ldots \times P(a_n = v_n \mid c_i) \quad (2)$$

Calculate the **product** for each value of i from 1 to k and choose the **classification with the largest value**.

Understanding Naïve Bayes Classification

- Classifications: {c*1*= On Time, c*2* = Late, c*3* = Very Late, c*4* = Cancelled}
- Prior Probabilities: P(On Time) = 0.7; P(Late) = 0.1; P(Very Late) = 0.15; P(Cancelled) = 0.05;.
- Attributes: {Day, Season, Wind, Rain}
- Values: . . .
- Calculate the product for each value of i from 1 to k and choose the classification that has the largest value

Method 3: Using Bayes

Searching for: {weekday,winter,high,heavy}, Using Equation 2

- P(a_1=v_1 | c_1)) means probability it's a weekday given that the class is ontime = 0.64
- P(a_2=v_2 | c_1)) means probability it's winter given that the class is ontime = 0.14
- P(a_3=v_3 | c_1)) means probability wind is high given that the class is ontime = 0.29
- P(a_4=v_4 | c_1)) means probability rain is heavy given that the class is ontime = 0.07
- **Posterior probability** for class = *on time*: 0.70 x 0.64 x 0.14 x 0.29 x 0.07 = 0.0013

Final Classifications

Searching for: {weekday,winter,high,heavy}?

- Calculate for c1 (class = ontime)?
  - 0.70 x 0.64 x 0.14 x 0.29 x 0.07 = 0.0013
- Calculate for c2 (class = late)?
  - 0.10 x 0.50 x 1.00 x 0.50 x 0.50 = 0.0125
- Calculate for c3 (class = very late)?
  - 0.15 x 1.00 x 0.67 x 0.33 x 0.67 = 0.0222
- Calculate for c4 (class = cancelled)?
  - 0.05 x 0.00 x 0.00 x 1.00 x 1.00 = 0.0000

Issues 1

- Note the 4 calculations are *not* probabilities: they do not add to 1.
- We could normalise so that all values add to 1, but we are only interested in finding the largest value.
- That is what we mean by *posterior probability of class c . . . be proportional to . . .*
- Issue: This method works but requires all attributes to be categorical (while some may be continuous)

Issues 2

- A second problem is that estimating probabilities by relative frequencies can give a poor estimate if the number of instances with a given attribute/value combination is small.
- In the extreme case where it is zero, the posterior probability will inevitably be calculated as zero.
- This happened for *class = cancelled* in the above example.
- This problem can be overcome by using a more complicated formula for estimating probabilities ...

Nearest Neighbour Overview

- Nearest Neighbour (NN) classification is generally used when all attributes are *continuous*.
- Concept: estimate the classification of an unseen instance using the classification of instances *closest* to it.

Table 1: 6 attributes and class: classify the 3rd instance!

a	b	c	d	e	f	class
yes	no	no	6.4	8.3	low	negative
yes	yes	yes	18.2	4.7	high	positive
yes	no	no	6.6	8.0	low	???

k-NN Classification

- We do not know what the 6 attributes represent but the answer appears obvious!
- We can reasonably predict its classification as *negative*.
- In practice, there will be many instances in a training set but the same principle applies ...
- Classification is based on the k nearest neighbours (k is generally small eg. 3 or 5), not just the (single) nearest one.
- This method is known as the *k-Nearest Neighbour* (or *k-NN*) classification.

k-NN Classification Algorithm

1. Locate the *k* training instances that are closest to the *unseen* instance
2. Take the most commonly occurring classification for these *k* instances.

Example Using *k-NN* Classification

How can we estimate the classification for an *unseen* instance with attributes 9.1 and 11.0 respectively?

Attribute 1	Attribute 2	Class
0.8	6.3	−
1.4	8.1	−
2.1	7.4	−
2.6	14.3	+
6.8	12.6	−
8.8	9.8	+
9.2	11.6	−
10.8	9.6	+
11.8	9.9	+
12.4	6.5	+
12.8	1.1	−
14.0	19.9	−
14.2	18.5	−
15.6	17.4	−
15.8	12.2	−
16.6	6.7	+
17.4	4.5	+
18.2	6.9	+
19.0	3.4	−
19.6	11.1	+

Figure 5: Training Set with two attributes and associated class

k=20 training instances

Consider the training set as 20 points (x,y) on a 2-D graph: points are + or - to indicate positive or negative class

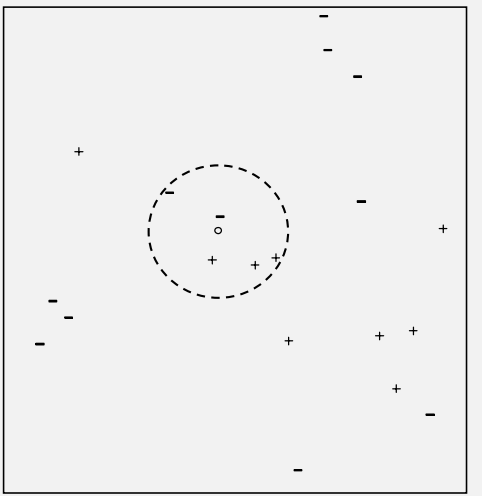


Figure 6: 2-D Representation of training data in Fig. 5

Calculating probability using a 5-*NN* classifier

- A circle encloses the 5 nearest neighbours of the unseen instance
- There are 3 + signs and 2 - signs
- The *5-NN* classifier classifies the unseen instance as *positive* (by 3 to 2)
- This is a 2-dimensional example and with 3 attributes, it becomes a 3 dimensional plane with (x,y,z) points.
- The number of attributes can increase rapidly: difficult to visualise but easily caculated using an algorithm.
- For *n* attributes, we represent the instances as points (a*1*,a*2*,..,a*n*) and (b*1*,b*2*,..,b*n*) in *n*-dimensional space.

Distance Measures

How do we compute the distance between 2 points?

- Goal: to measure the distance between two instances with n attributes (or in n-space).
- Notation: **dist**(X,Y) can be used to denote the distance between two points X and Y in n-space.
- There are 3 conditions:
  1. The distance of any point from itself is zero: **dist**(A,A) = 0.
  2. The distance from A to B and from B to A are identical: **dist**(A,B) = **dist**(B,A)
  3. The shortest distance between any 2 points is a straight line.

The Triangle Inequality

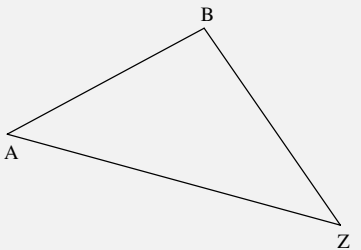


Figure 7: The Triangle Inequality

- The third condition is the *triangle inequality.*
- It means that the shortest distance between any 2 points is a straight line.
- For any points *A,B,Z*: **dist**(A,B) $\leq$ **dist**(A,Z) + **dist**(Z,B)
- Equality occurs only if Z=A or Z=B or Z is on the direct route between A and B.

Euclidean Distance

- We start by illustrating the formula for Euclidean distance in 2-D.
- Assume an instance in the training set to be $u = (a_1, a_2)$ and the unseen instance by $v = (b_1, b_2)$.
- Then the length of the straight line joining the points is:

If $u = (a_1, a_2)$ and $v = (b_1, b_2)$ are two points on the plane, their *Euclidean distance* is given by

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \tag{3}$$

Euclidean Distance for 3-dimensions

- We can now illustrate the formula for Euclidean distance in 3-D.
- Assume an instance in the training set to be $u = (a_1, a_2, a_3)$ and the unseen instance by $v = (b_1, b_2, b_3)$.

If $u = (a_1, a_2, a_3)$ and $v = (b_1, b_2, b_3)$ are two points in three dimensional space the corresponding formula is

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}. \quad (4)$$

Euclidean Distance for *n*-dimensions

If $u = (a_1, a_2, .., a_n)$ and $v = (b_1, b_2, .., b_n)$ are two points in *n*-dimensional space the corresponding formula is

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}. \quad (5)$$

- In data mining terms, you can now classify any unseen (new) instance, with any number of attributes, using a set of known instances.

Normalisation

Large values can skew the Euclidean distance formula

Mileage	No. of doors	Age (years)	Owners
18,457	2	12	8
26,292	4	3	1

- Assume the above classification problem for cars.
- When the distance to an unseen instance is calculated, the mileage attribute will contribute a very large value.
- Mileage will be the dominant attribute.
- It is the chosen *unit of measure* that causes the problem.
- Other units could have been chosen, eg. for age in seconds, leading to similar problems.

Solution

- Normalise all attributes so they run from 0 to 1.
- Assume a range of -8.1 to 94.3. Add 8.1 to all values and range becomes 0 to 102.4. Then divide all by 102.4.
- In general terms, for each attribute *a*, we apply the formula:

$$\frac{a - min}{max - min} \tag{6}$$

If an unseen instance arrives which is outside the range *{min,max}* it is necessary to calibrate all values again!

Weightings

- Where attributes are not considered equal, we may assign weightings to each.
- Note: it is normal to scale the weights so that the sum of all weights = 1.

The Euclidean formal with weightings becomes:

$$\sqrt{w_1(a_1 - b_1)^2 + w_2(a_2 - b_2)^2 + w_n(a_n - b_n)^2}. \qquad (7)$$

Decision Trees

- **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** (non-leaf 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**.

Decision Tree for the concept *buys_computer*

- The tree predicts whether a customer at *AllElectronics* is likely to purchase a computer.
- Internal nodes are denoted by rectangles and leaf nodes are denoted by ovals.

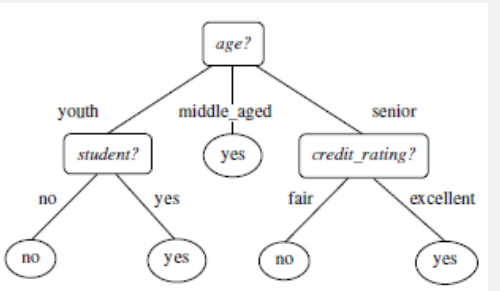


Figure 8: Is Customer likely to buy a Computer?

Decision Rules and Decision Trees

- One method of constructing a model from a dataset is to construct a *decision tree*.
- This can also be seen as a *set of decision rules*.
- A well used example, is that of the golfer who *makes a decision* to play, based on the weather.
- Based on the dataset in table 9, would the golfer play if attributes were:
  {sunny, 74F, 77% humidity, false (windy)}?

Classify: {sunny, 74F, 77% humidity, false (windy)}

Outlook	Temp (°F)	Humidity (%)	Windy	Class
sunny	75	70	true	play
sunny	80	90	true	don't play
sunny	85	85	false	don't play
sunny	72	95	false	don't play
sunny	69	70	false	play
overcast	72	90	true	play
overcast	83	78	false	play
overcast	64	65	true	play
overcast	81	75	false	play
rain	71	80	true	don't play
rain	65	70	true	don't play
rain	75	80	false	play
rain	68	80	false	play
rain	70	96	false	play

Classes
play, don't play
Outlook
sunny, overcast, rain
Temperature
numerical value
Humidity
numerical value
Windy
true, false

Figure 9: Data for Golf Example

Building the Decision Tree

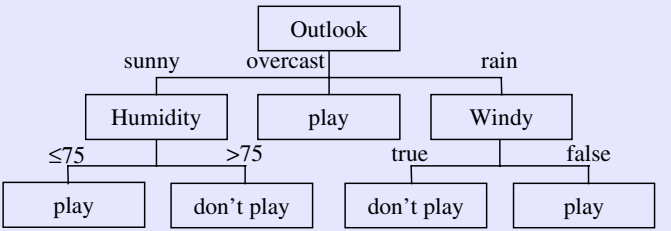


Figure 10: Decision Tree for Golf Example

- Begin with the value for *Outlook* and there are 3 possibilities.

Building the Decision Tree

- Begin with the value for *Outlook* and there are 3 possibilities.
  1. If *Outlook* is *sunny*, consider *Humidity*. If the value is ≤ 75, the decision (class) is *play*, otherwise *don't play*
  2. If *Outlook* is *overcast*, the decision (class) is *play*.
  3. If *Outlook* is *rain*, consider *Windy*. If the value is *true*, the decision (class) is *don't play*, otherwise *play*.
- Note that *Temperature* is never used.

Terminology

- There is a universe of *objects*, each described by its *attributes*.
- Attributes with a finite (and typically small) set of values are called *categorical*.
- Attributes with numerical values are generally known as *continuous*.
- One distinguishes between a special form of categorical attribute called the *classification* and all other attributes.
- Descriptions of sample objects are held in tabular form known as a *training set*.
- Each row is an *instance*, comprising attribute values and a classification.

Decision Tree Algorithm

- The tree starts as a single node N, representing the *training tuples* in D.
- If the tuples in D are all of the same class, then node N becomes a **leaf** and is *labeled* with that **class**.
- Otherwise, the algorithm calls an `AttributeSelection` 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.
- It also tells us which branches to *grow* from node N, determined by the outcomes of the chosen test.
- This splitting attribute may also indicate either a *split-point* or a *splitting subset*.

Decision Tree Algorithm: Splitting

- The splitting criterion is determined so resulting partitions at each branch are as *pure* as possible.
- A partition is *pure* if all of its tuples belong to the same class.
- If we split tuples in D according to the mutually exclusive outcomes of the splitting criterion, we expect the resulting partitions to be as pure as possible.
- The node N is labeled with the splitting criterion, which serves as a test at the node.
- A branch is *grown* from node N for *each* outcome of the splitting criterion.
- The tuples in D are partitioned accordingly.
- There are 3 possibilities for the splitting attribute A.

Partitioning Scenarios

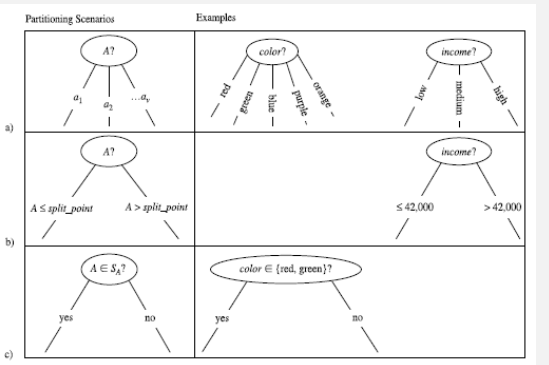


Figure 11: 3 Possibilities for Partitioning Tuples

1. A is discrete-valued

- Here, the outcomes of the test at node N correspond directly to the known values of A.
- A branch is created for **each known value** a_j, of A and labeled with that value (Figure 11(a)).
- Partition D_j is the subset of class-labeled tuples in D having value a_j of A.
- As all tuples in a given partition have the same value for A, A need not be considered in any future partitioning.

2. A is continuous-valued

- In this case, the test at node N has two possible outcomes corresponding to the conditions: $A \leq$ `split-point` and $A >$ `split-point` respectively, where `split-point` is the split-point returned by `AttributeSelection` method as part of the splitting criterion.
- The split-point a, can be taken as the midpoint of two known adjacent values of A.
- Two branches are grown from N and labeled according to the outcomes in Figure 11(b).
- Tuples are partitioned such that D_1 holds the subset of class-labeled tuples in D for which $A \leq$ `split-point`, while D_2 holds the rest.

3. A is discrete-valued and a binary tree must be produced

- The test at node N is of the form: $A \in S_A$?.
- S_A is the splitting subset for A, returned by `AttributeSelection` method as part of the splitting criterion.
- It is a subset of the known values of A.
- If a given tuple has value a_j of A and if $a_j \in S_A$, then the test at node N is satisfied.
- Two branches are grown from N (Figure 11(c)).
- By convention, the left branch of N is labeled *yes* so that D_1 corresponds to the subset of class-labeled tuples in D, that satisfy the test.
- The right branch out of N is labeled *no* so that D_2 corresponds to the subset of class-labeled tuples from D, that do not satisfy the test.

Terminating Conditions

- The algorithm uses the same process recursively to form a decision tree for tuples at each resulting partition D_j of D.
- The recursive partitioning stops when one of the following terminating conditions is true:
  1. All of the tuples in partition D (represented at node N) belong to the same class.
  2. There are no remaining attributes on which the tuples may be further partitioned.
     In this case, majority voting is employed which involves converting node N into a leaf and labeling it with the most common class in D.
  3. There are no tuples for a given branch: a partition D_j is empty.
     In this case, a leaf is created with the majority class in D.

Decision Tree Terminology

The goal is to develop *classification rules* from training set data where a *decision tree* is formed.

- A decision tree is constructed by *splitting on* (the value of) *attributes*.
- This involves testing the value of an attribute (eg. *Outlook*) and creating a branch for each possible value (*sunny, overcast, rain*).
- For continuous attributes, the test is generally using $<$, $<=$, $>$ etc. using a *split value*.
- The splitting process continues until each branch can be labelled with only a single classification.

Decision Tree Functions

- Decision trees have two different functions: *data compression* and *prediction*.
- Figure 10 could be regarded as a compact representation of Figure 9.
- Both representations are equivalent: for each of the 14 instances, the given values will lead to identical classifications.
- However, the decision tree can also be used for *prediction* (for values not in the dataset).
- Would the golfer play if attributes were: {sunny, 74F, 77% humidity, false}?

Decision Tree Functions

- The prediction is NO (but this is only a prediction!)
- Thus, the decision tree is not merely equivalent to the original training set but is also a *generalisation* which can be used to predict the classification of *unseen instances*.
- A collection of unseen instances is known as a *test set* or *unseen test set*.

Degrees Dataset Example

SoftEng	ARIN	HCI	CSA	Project	Class
A	B	A	B	B	SECOND
A	B	B	B	A	FIRST
A	A	A	B	B	SECOND
B	A	A	B	B	SECOND
A	A	B	B	A	FIRST
B	A	A	B	B	SECOND
A	B	B	B	B	SECOND
A	B	B	B	B	SECOND
A	A	A	A	A	FIRST
B	A	A	B	B	SECOND
B	A	A	B	B	SECOND
A	B	B	A	B	SECOND
B	B	B	B	A	SECOND
A	A	B	A	B	FIRST
B	B	B	B	A	SECOND
A	A	B	B	B	SECOND
B	B	B	B	B	SECOND
A	A	B	A	A	FIRST
B	B	B	A	A	SECOND
B	B	A	A	B	SECOND
B	B	B	B	A	SECOND
B	A	B	A	B	SECOND
A	B	B	B	A	FIRST
A	B	A	B	B	SECOND
B	A	B	B	B	SECOND
A	B	B	B	B	SECOND

Classes
FIRST, SECOND
SoftEng
A,B
ARIN
A,B
HCI
A,B
CSA
A,B
Project
A,B

Figure 12: The *degrees* Dataset

The degrees Example

- What determines a classification of 1st or 2nd?
- Figure 12 shows a decision tree where every branch has a (*leaf node* with a) possible decision.
- Each branch comprises a route from *root* to leaf.
- A node that is neither root nor leaf is an *internal node*.

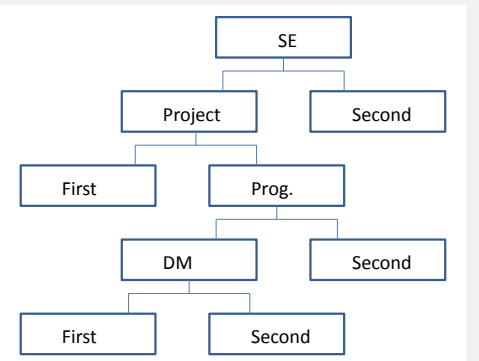


Figure 13: Degrees Decision Tree

The degrees Example

Each branch corresponds to a classification rule

1. If SE = *A* and Project = *A* then Class = *First*
2. If SE = *A* and Project = *B* and Prog. = *A* and DM = *A* then Class = *First*
3. If SE = *A* and Project = *B* and Prog. = *A* and DM = *B* then Class = *Second*
4. If SE = *A* and Project = *B* and Prog. = *B* then Class = *First*
5. If SE = *B* then Class = *First*

Decision Rules: Reduction

- The left hand side of each rule (the *antecedent*) comprises a number of *terms* joined by the AND operator.
- **Compression**: the decision tree has 5 decision rules, with a total 14 terms, an average of 2.8 terms per rule.
- Each instance in the Degrees dataset could also be regarded as a rule eg. If SE = *A* and Prog = *B* and HCI = *A* and DM = *B* and Project = *B* then Class = *Second*
- There are 26 such rules, one per instance, each with 5 terms (total = 130).
- Even with a small training set, the reduction from 130 (fig. 12) to 14 (fig. 13) terms is almost 90%.

TDIDT Algorithm

- Decision trees are widely used as a means of generating classification rules because of the existence of a simple but very powerful algorithm called TDIDT, which stands for **Top-Down Induction of Decision Trees**.
- The method produces decision rules in the implicit form of a decision tree.
- Decision trees are generated by repeatedly splitting on the values of attributes.
- This process is known as recursive partitioning.

Basic TDIDT Algorithm

TDIDT: BASIC ALGORITHM

IF all the instances in the training set belong to the same class
THEN return the value of the class
ELSE (a) Select an attribute A to split on$^+$
(b) Sort the instances in the training set into subsets, one for each value of attribute A
(c) Return a tree with one branch for each *non-empty* subset, each branch having a descendant subtree or a class value produced by applying the algorithm recursively

$^+$ Never select an attribute twice in the same branch

Figure 14: Basic TDIDT Algorithm

TDIDT Algorithm

- In the standard formulation of the TDIDT algorithm there is a training set of instances.
- Each instance corresponds to a member of a universe of objects, which is described by the values of a set of categorical attributes.
- The basic algorithm can be given in just a few lines as shown in Figure 14.

TDIDT Algorithm Discussion (1)

- At each non-leaf node an attribute is chosen for splitting.
- This can potentially be any attribute, except that the same attribute must not be chosen twice in the same branch.
- This restriction is innocuous: in the branch corresponding to the incomplete rule
  IF `SoftEng` = *A* AND `Project` = *B* . . .
  it is not permitted to choose `SoftEng` or `Project` as the next attribute to split on, but as their values are already known there would be no point in doing so.

TDIDT Algorithm Discussion (2)

- However this harmless restriction has a very valuable effect.
- Each split on the value of an attribute extends the length of the corresponding branch by one term, but the maximum possible length for a branch is M terms where there are M attributes.
- Thus, the algorithm is guaranteed to terminate.

TDIDT Algorithm: Condition for Usage

- There is one important condition which must hold before the TDIDT algorithm can be applied.
- This is the Adequacy Condition:
  No two instances with the **same values for all the attributes** may belong to **different** classes.
- This is simply a way of ensuring that the training set is consistent.

TDIDT Algorithm: Underspecification

- A major problem with the TDIDT algorithm, which is not apparent at first sight, is that it is *underspecified*.
- The algorithm specifies 'Select an attribute A to split on' but no method is given for doing this.
- Provided the adequacy condition is satisfied, the algorithm is guaranteed to terminate and any selection of attributes (even random selection) will produce a decision tree, provided that an attribute is never selected twice in the same branch.

TDIDT Issue: Attribute Selection

- Is under-specification a problem?
- It might be! Many of the decision trees (and the corresponding decision rules) will be of little value for predicting the classification of unseen instances.
- Thus, some methods of selecting attributes may be much more useful than others.
- A good choice of attributes to split on at each stage is crucial to the success of the TDIDT approach.

Attribute Selection

- Using TDIDT and provided that the adequacy condition is satisfied, any method of choosing attributes will produce a decision tree.
- It is important to understand the decision trees obtained from using some poorly chosen strategies for attribute selection.
- We can then describe one of the most widely used approaches and look at how the results compare.

Selection Strategies

Different decision trees are produced by using the three attribute selection strategies listed below.

1. `takefirst`. For each branch take the attributes in the order in which they appear in the training set, working from left to right, e.g. for the degrees training set in the order SoftEng, Prog, HCI, DM and Project.
2. `takelast`. As for takefirst, but working from right to left, e.g. for the degrees training set in the order Project, DM, HCI, Prog, and SoftEng.
3. `random`. Make a random selection (with equal probability of each attribute being selected).

Selection Strategies

1. As always no attribute may be selected twice in the same branch (See figure 15 results.)
2. TDIDT algorithm with attribute selection strategies `takefirst`, `takelast` and `random` in turn to generate decision trees for the seven datasets: *contact lenses, lens24, chess, vote, monk1, monk2 and monk3*.
3. The random strategy was used five times for each dataset.
4. In each case, the value given in the table is **the number of branches** in the decision tree generated.

Decision Tree Strategies: Results

Dataset	take first	take last	random					most	least
			1	2	3	4	5		
contact_lenses	42	27	34	38	32	26	35	42	26
lens24	21	9	15	11	15	13	11	21	9
chess	155	56	94	52	107	90	112	155	52
vote	40	79	96	78	116	110	96	116	40
monk1	60	75	82	53	87	89	80	89	53
monk2	142	112	122	127	109	123	121	142	109
monk3	69	69	43	46	62	55	77	77	43

Figure 15: Different Selection Attributes = Different Trees

Selection Strategies: Analysis

1. See figure 15 results.
2. The last two columns record the number of branches in the largest and the smallest of the trees generated for each of the datasets.
3. In all cases, there is a considerable difference.
4. This suggests that although in principle the attributes can be chosen in any arbitrary way, the difference between a good choice and a bad one may be considerable.

Football-Netball Example

- Students must enrol in either the Football Club or the Netball Club.
- It is forbidden to join both clubs.
- Figure 16 gives a training set of data collected about 12 students, tabulating four items of data about each one (eye colour, marital status, sex and hair length) against the club joined.
- What determines who joins which club?

Football-Netball Training Set

eyecolour	married	sex	hairlength	class
brown	yes	male	long	football
blue	yes	male	short	football
brown	yes	male	long	football
brown	no	female	long	netball
brown	no	female	long	netball
blue	no	male	long	football
brown	no	female	long	netball
brown	no	male	short	football
brown	yes	female	short	netball
brown	no	female	long	netball
blue	no	male	long	football
blue	no	male	short	football

Figure 16: Training Set for Football-Netball Example

Football-Netball Tree 1 Analysis (1)

- It is possible to generate many different trees from this data using the TDIDT algorithm.
- One possible decision tree is Figure 17.
- The numbers in parentheses indicate the number of instances corresponding to each of the leaf nodes.
- This is a remarkable result!
- All the blue-eyed students play football.

Football-Netball Decision Trees

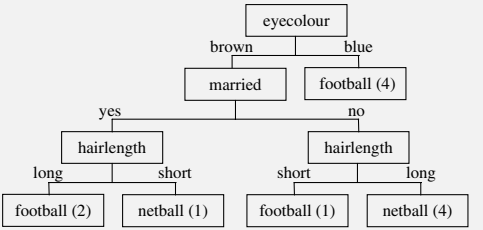


Figure 17: Decision tree a)

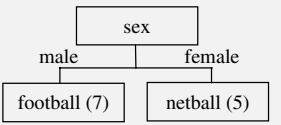


Figure 18: Decision tree b)

Football-Netball Tree Analysis (1)

- For the brown-eyed students, the critical factor is whether or not they are married.
- If they are, then the long-haired ones all play football and the short-haired ones all play netball.
- If they are not married, it is the other way round: the short-haired ones play football and the long-haired ones play netball.
- This would be an astonishing discovery, likely to attract worldwide attention, if it were correct: but is it?

Football-Netball Tree Analysis (2)

- Figure 18 looks more believable but is it correct?
- Although it is tempting to say that it is, it is best to avoid using terms such as *correct* and *incorrect* in this context.
- All we can say is that both decision trees are compatible with the data from which they were generated.
- The only way to know which one gives better results for unseen data is to use them both and compare the results.
- Despite this, it is hard to avoid the belief that Figure 18 is right and Figure 17 is wrong.

Anonymous Training Set

a1	a2	a3	a4	class
a11	a21	a31	a41	c1
a12	a21	a31	a42	c1
a11	a21	a31	a41	c1
a11	a22	a32	a41	c2
a11	a22	a32	a41	c2
a12	a22	a31	a41	c1
a11	a22	a32	a41	c2
a11	a22	a31	a42	c1
a11	a21	a32	a42	c2
a11	a22	a32	a41	c2
a12	a22	a31	a41	c1
a12	a22	a31	a42	c1

Figure 19: Real-World Data Mining Example

Anonymous Decision Trees

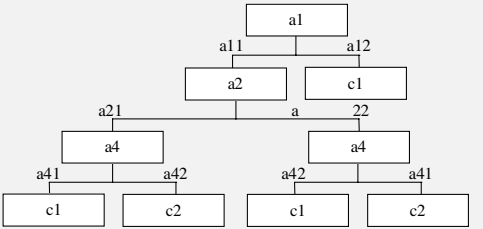


Figure 20: Decision tree a)

Figure 21: Decision tree b)

Anonymous Analysis: which tree is better? (1)

- Figure 21 because it is smaller, there seems no reason why a) should not be acceptable.
- Data mining algorithms generally do not allow the use of any background knowledge on the domain from which the data is drawn.
- Could be: meaning and relative importance of attributes, or which attributes are most or least likely, to determine the classification of an instance.
- How does this compare to the netball example?

Anonymous Analysis: which tree is better? (2)

- It is easy to see that a decision tree involving tests on *eyecolour*, *hairlength* etc. is meaningless when given in isolation.
- If those attributes were part of a much larger number (possibly many thousands) in a practical application how can one prevent meaningless decision rules?
- Apart from vigilance and a good choice of algorithm, the answer to this is *nothing*!
- The quality of the strategy used to *select the attribute to split on* at each stage is clearly of vital importance.

Attribute Selection Measures

- An attribute selection measure is a heuristic for selecting the splitting criterion that *best* separates data partition D of class-labeled training tuples into individual classes.
- If one splits D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be *pure*: all tuples that fall into a given partition belong to the same class.
- Conceptually, the *best* splitting criterion is the one that most closely results in such a scenario.
- Attribute selection measures are also known as *splitting rules* because they determine how to split the tuples at a given node.
- The attribute selection measure provides a *ranking* for each attribute describing the training tuples.
- The attribute having the best score for the measure is chosen as the splitting attribute.

Attribute Selection Measures

- If the splitting attribute is *continuous* or if we are restricted to binary trees, then either a split point or a splitting subset (respectively) must also be determined as part of the splitting criterion.
- The tree node created for partition D is labeled with the splitting criterion; branches are grown for each outcome of the criterion; and the tuples are partitioned accordingly.
- We know cover three popular attribute selection measures: **information gain**, **gain ratio**, and **gini index**.

Fundamentals

Let data partition D be a training set of class-labeled tuples.
Assume the class label attribute has m distinct values defining m distinct classes C_i ($i = 1,2,..,m$).
Let $C_{i,D}$ be the set of tuples of class C_i in D.
Let $| D |$ *and* $| C_{i,D} |$ denote the number of tuples in D and $C_{i,D}$ respectively.

Information Gain

- **Information Gain** uses the value or *information content* of messages.
- Let node *N* represent 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 **entropy** in these partitions.
- Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

Information Gain Equation *Info(D)*

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

$$Info(D) = -\sum_{i=1}^{m} p_i \ log_2 \ p_i \qquad (8)$$

- where p_i is the probability that an arbitrary tuple in D belongs to class C_i and is estimated by $| \ C_{i,D} \ |/| \ D \ |$
- A log function to the base 2 is used, because the information is encoded in bits.
- *Info(D)* is just the *average* amount of information needed to identify the class label of a tuple in *D*.

Partitioning & Purity

- Assume we must partition the tuples in D on some attribute A having v **distinct** values $\{a_1,a_2,..,a_v\}$ as observed from the training data.
- If A is discrete-valued, these values correspond directly to the v outcomes of a test on A.
- Attribute A can be used to split D into v partitions or subsets, $\{D_1,D_2,..,D_v\}$ where D_j contains those tuples in D that have outcome a_j of A.
- These partitions would correspond to the branches grown from node N.
- Ideally, we would like this partitioning to produce an exact classification of the tuples: each partition to be pure!
- However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class).

Exact Classification

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

$$Info_A(D) = \sum_{j=1}^{v} \frac{\mid D_j \mid}{\mid D \mid} \times Info(D_j) \qquad (9)$$

- $\text{Info}_A(\text{D})$ is the expected information required to classify a tuple from D based on the partitioning by A.
- The smaller the expected information (still) required, the greater the purity of the partition.
- The term $\frac{|D_j|}{|D|}$ acts as the weight of the jth partition.

Information Gain: Equation

Information gain is defined as the *difference* between the original information requirement (based on the proportion of classes) and the new requirement (obtained *after* partitioning on A). That is:

$$Gain(A) = Info(D) - Info_A(\mathrm{D}) \tag{10}$$

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 node N.

AllElectronics Training Set

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

Figure 22: Class-Labelled Training Tuples

Example using Figure 22 dataset

- In Figure 22, *D* is class-labeled tuples randomly selected from the *AllElectronics* customer database.
- Each attribute is discrete-value and continuous-valued attributes have been generalized.
- The class label attribute *buys_computer*, has two distinct values {yes, no} and thus, m = 2.
- Let class C_1 correspond to *yes* and class C_2 correspond to *no*.
- There are 9 tuples of class *yes* and 5 tuples of class *no*.
- A (root) node *N* is created for the tuples in *D* and to find the splitting criterion, compute the **information gain** of each attribute.

Initial Classification

- We first use Equation 10 to compute the expected information needed to classify a tuple in D:

$$Info(D) = -\frac{9}{14} log_2\left(\frac{9}{14}\right) - \frac{5}{14} log_2\left(\frac{5}{14}\right) = 0.94 bits$$

- Next, compute the expected information requirement for each attribute.

Start with the attribute *age*

- Examine the distribution of *yes* and *no* tuples for each category of *age*.
- For the *age* category *youth*: there are 2 *yes* and 3 *no* tuples.
- For the category *middle aged*: there are 4 *yes* and 0 *no* tuples.
- For the category *senior*: there are 3 *yes* and 2 *no* tuples.

Partitioning by Age

- Using Equation 9 the expected information needed to classify a tuple in D if the tuples are partitioned according to *age* is

$$\begin{aligned} Info_{age}(D) = & \frac{5}{14} \times (-\frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5}) \\ & + \frac{4}{14} \times (-\frac{4}{4} log_2 \frac{4}{4} - \frac{0}{4} log_2 \frac{0}{4}) \\ & + \frac{5}{14} \times (-\frac{3}{5} log_2 \frac{3}{5} - \frac{2}{5} log_2 \frac{2}{5}) \\ & = 0.694 bits \end{aligned}$$

- Thus, the gain in information from this partitioning is:

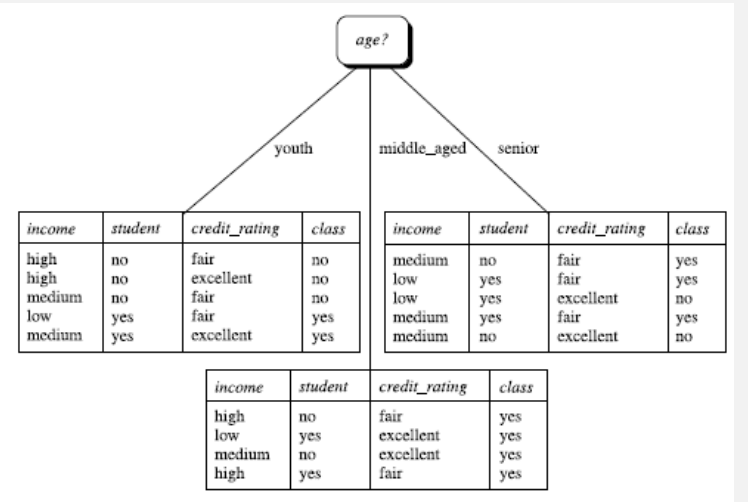
$$\begin{aligned} Gain(age) &= Info(D) - Info_{age}(D) \\ &= 0.94 - 0.69 = 0.246 bits \end{aligned}$$

Splitting by Remaining Attributes

Similarly, we can compute:
$Gain(income) = 0.029 bits$;
Gain(student) = 0.151 bits;
Gain(credit_rating) = 0.048 bits

- Because *age* has the highest information gain among the attributes, it is chosen as the splitting attribute.
- Node N is labeled with *age*, and branches are grown for each of the attribute's values.
- The tuples are then partitioned accordingly, as shown in Figure 23.

AllElectronics Training Set



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

Figure 23: As Age has the highest Information Gain, it becomes the splitting attribute

Final Decision Tree

- Note that the tuples falling into the partition for *age = middle aged* all belong to the same class.
- Because they all belong to class *yes*, a leaf should thus be created at the end of this branch and labeled with *yes*.
- The final decision tree returned by the algorithm is shown in Figure 24.

Final Decision Tree: Figure

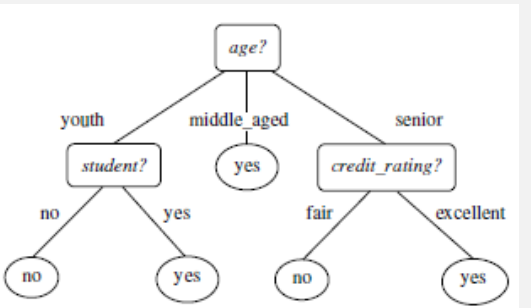


Figure 24: Final Decision Tree

Continuous-Valued Attributes

- Assume we have an attribute A that is *continuous-valued* rather than discrete-valued.
- Instead of the discretised version of *age*, we instead have the raw values for this attribute.
- For such a scenario, we must determine the *best* split-point for A.
- We first sort the values of A in increasing order.

Calculate split-point

- In general, the midpoint between each pair of adjacent values is considered as a possible split-point.
- Thus, given *v* values of *A*, then *v-1* possible splits are evaluated.
- For example, the midpoint between the values a_i and a_{i+1} of A is:

$$\frac{a_i + a_{i+1}}{2}$$

Calculate split-point

- If the values of A are sorted in advance, then determining the best split for A requires only one pass.
- For each possible split-point for A, we evaluate $\text{Info}_A(D)$, where the number of partitions is two: $v = 2$ (or $j = 1,2$) in Equation 9.
- The point with the minimum expected information requirement for A is selected as the split point for A.
- D_1 is the set of tuples in D satisfying $A \leq$ `split-point`,
  and D_2 is the set of tuples satisfying $A >$ `split-point`.

Gain Ratio

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

Gain Ratio

- An extension to information gain known as **gain ratio**, attempts to overcome this bias.
- It applies a form of normalization to information gain using a *split information* value defined analogously with *Info(D)* as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{\mid D_j \mid}{\mid D \mid} \times log_2 \left(\frac{\mid D_j \mid}{\mid D \mid} \right) \qquad (11)$$

SplitInfo explained

- The `SplitInfo` 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.
- For *each outcome*, it considers the **number of tuples** having that outcome with respect to the total number of tuples in D.
- It differs from information gain, which measures the information with respect to **classification** that is acquired based on the same partitioning.

Gain Ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \tag{12}$$

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

Example for attribute *Income*

- A test on *income* splits the data in figure 22 into three partitions: *low*, *medium*, and *high*, containing 4, 6, and 4 tuples respectively.
- To compute the gain ratio of *income*, we first use Equation 11 to obtain:

$$\begin{aligned} SplitInfo_A(D) = &-\frac{4}{14} \times log_2\left(\frac{4}{14}\right) \\ &-\frac{6}{14} \times log_2\left(\frac{6}{14}\right) \\ &-\frac{4}{14} \times log_2\left(\frac{4}{14}\right) \\ &= 0.926 \end{aligned}$$

We have *Gain(income) = 0.029*.
Therefore, *GainRatio(income) = 0.029/0.926 = 0.031*.

Gini Index

- The Gini index measures the impurity of D, a data partition or set of training tuples

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{13}$$

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by
$| C_{i,D} | / | D |$
The sum is computed over m classes.

Binary Split

- The Gini index considers a *binary* split for each attribute.
- First consider the case where A is a discrete-valued attribute having v distinct values $\{a_1, a_2, .., a_v\}$ in D.
- To determine the best binary split on A, examine all possible subsets that can be formed using known values of A.
- Each subset S_A, can be considered as a binary test for attribute A of the form: $A \in S_A$?.
- Given a tuple, this test is satisfied if the value of A for the tuple is among the values listed in S_A.
- If A has v possible values, then there are 2^v possible subsets.

Gini Example

- For example, if *income* has three possible values, namely *{low, medium, high}*, then the possible subsets are *{low, medium, high}*, *{low, medium}*, *{low, high}*, *{medium, high}*, *{low}*, *{medium}*, *{high}*, *and {}*.
- We exclude the power set *{low, medium, high}*, and the empty set from consideration since conceptually, they do not represent a split.
- Therefore, there are 2^v-2 possible ways to form two partitions of the data *D*, based on a binary split on *A*.
- When considering a binary split, we compute a weighted sum of the impurity of each resulting partition.

Gini Index: discrete values

- For example, if a binary split on A partitions D into D_1 and D_2, the gini index of D given that partitioning is:

$$Gini_A(D) = \frac{\mid D_1 \mid}{\mid D \mid} Gini(D_1) + \frac{\mid D_2 \mid}{\mid D \mid} Gini(D_2) \qquad (14)$$

- For each attribute, every possible binary split is considered.
- For a *discrete-valued* attribute, the subset that gives the *minimum* gini index for that attribute is selected as its splitting subset.

Gini Index: continuous values

- For *continuous-valued* attributes, each possible split-point must be considered.
- The strategy is similar to that of **information gain**, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point.
- The point giving the minimum Gini index for a given (continuous-valued) attribute is taken as the split-point of that attribute.
- Recall that for a possible split-point of A, D_1 is the set of tuples in D satisfying $A \leq$ `split-point`, and D_2 is the set of tuples in D satisfying $A >$ `split-point`.

Reduction in Impurity

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

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

- The attribute that maximizes the reduction in impurity (has the minimum Gini index) is selected as the splitting attribute.
- This attribute and either its splitting subset (for a discrete-valued splitting attribute) or split-point (for a continuous valued splitting attribute) together form the splitting criterion.

Induction of a decision tree using gini index

- Let D be the training data in figure 22, where there are 9 tuples belonging to the class *buys computer = yes* and the remaining 5 tuples belong to the class *buys computer = no*.
- A (root) node N is created for the tuples in D.
- We first use Equation 13 for Gini index to compute the impurity of D

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

Starting with *income*

- To find the splitting criterion for the tuples in D, we need to compute the gini index for each attribute.
- Start with the attribute *income* and consider each of the possible splitting subsets.
- Consider the subset *{low, medium}*.
- This would result in 10 tuples in partition D_1 satisfying the condition: $income \in \{low, medium\}$.
- The remaining 4 tuples of D would be assigned to partition D_2.

Gini Index for *income*

- The Gini index value based on this partitioning is:

$$
\begin{aligned}
Gini_{income \in \{low,\, medium\}^{(D)}} &= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\
&= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\
&= 0.45 \\
&= Gini_{income \in \{high\}^{(D)}}
\end{aligned}
$$

Remaining Gini Values

- Similarly, the Gini index values for splits on the remaining subsets are:
  0.315 for the subsets *{low, high}* and *{medium}*; and
  0.300 for the subsets *{medium, high}* and *{low}*.
- Therefore, the best binary split for attribute *income* is on *{medium, high}* (or *{low}*) because it minimizes the gini index.
- Evaluating the attribute, we obtain *{youth, senior}* (or *{middle aged}*) as the best split for *age* with a Gini index of 0.375;
  the attributes *{student}* and *{credit rating}* are both binary, with Gini index values of 0.367 and 0.429 respectively.

Gini selects *income*

- The attribute *income* and splitting subset *{medium, high}* therefore give the minimum gini index overall, with a reduction in impurity of 0.459-0.300 = 0.159.
- The binary split $income \in \{medium, high\}$ results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion.
- Node N is labeled with the criterion, two branches are grown from it, and the tuples are partitioned accordingly.
- Hence, the Gini index has selected *income* instead of *age* at the root node, unlike the (non-binary) tree created by the **information gain** example.

Conclusions

- Our examination of attribute selection measures is not exhaustive but we focused on the three measures that are commonly used for building decision trees.
- These measures are not without their biases.
  1. Information gain is biased toward multivalued attributes.
  2. Although the gain ratio adjusts for this bias, it tends to prefer unbalanced splits in which one partition is much smaller than the others.
  3. The Gini index is biased toward multivalued attributes and has difficulty when the number of classes is large.
     It also tends to favor tests that result in equal-sized partitions and purity in both partitions.
- Although biased, these measures give reasonably good results in practice.