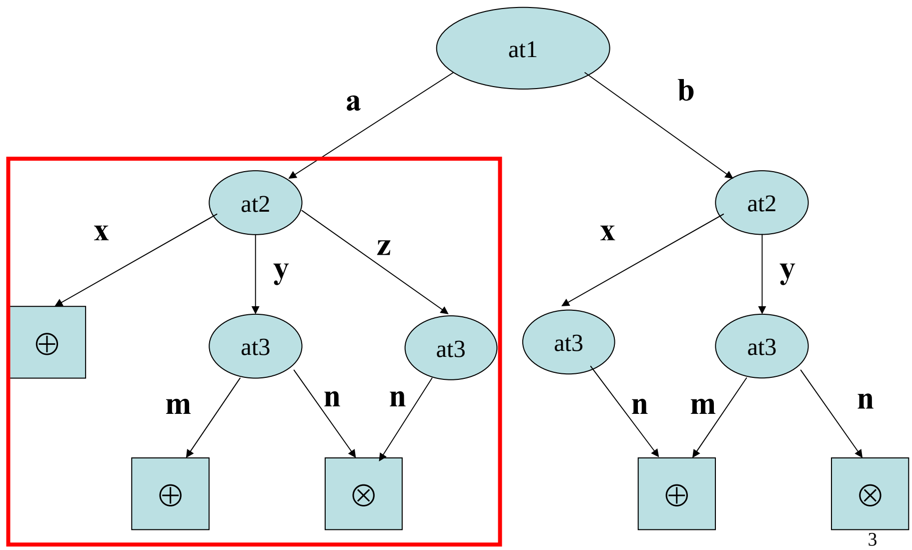
Data mining, machine learning, and uncertainty reasoning

林偉川

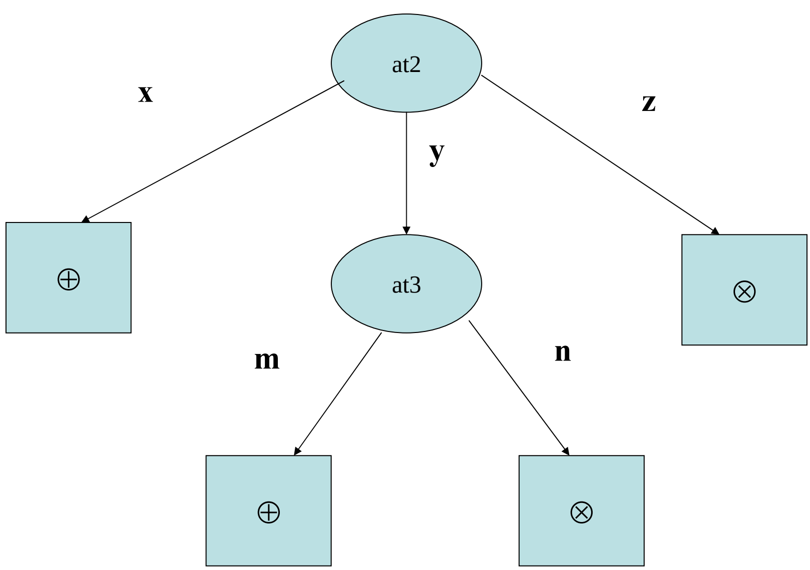
Positive and negative examples for concept learning

example	at1	at2	at3	Classification
e1	a	x	n	⊕ (positive)
e2	b	x	n	⊕
e3	a	y	n	⊗ (negative)
e4	a	z	n	⊗
e5	a	y	m	⊕
e6	b	y	n	⊗
e7	b	y	m	⊕
e8	a	x	m	⊕

Decision tree of example table



Decision tree of example table



TDIDT Algorithm

S … the set of examples

1. Find the "best" attribute *at* ***(if this can be found!!!)***
2. Split the set S into the subset S_1, S_2, …, so that all examples in the subset S_i have $at=v_i$. Each subset constitutes a node in the decision tree
3. For each S_i : if all examples in S_i belong to the same class (⊗ or ⊕), then create a leaf of the decision tree and label it with this class label (such as x or z) . Otherwise, perform the same procedure (go to step 1) with S=Si (such as *at3*)

TDIDT Algorithm

- The entire set of examples is split into subsets that are more easy to handle
- With a properly defined evaluation function, the TDIDT algorithm will derive the proper decision tree
- This Divide-and-conquer algorithm terminates when all subsets are labeled or when no further attributes splitting the unlabelled sets are available
- Complete the full decision tree can cover the table example but they include some negative examples
- The most difficult is how to find the first best attribute in step 1!!

How to find the best attribute

- The evaluation function that is satisfied the following requirements:
  - The function reaches its maximum when all subsets are homogeneous ➔ all examples in S_i are ⊗ or all examples in S_i are ⊕, the information about the attribute value is sufficient to decide whether the example is positive or negative
  - The function reaches its minimum when 50% of the examples in each of the subsets are positive and 50% are negative
  - The function should be steep when close to the extremes (100% positive or vice versa) and flat when in the 50%-50% region

Entropy

- Information is maximized when another important quantity "entropy" is minimized
- Entropy determines the extent of randomness, "un-structuredness", chaos in the data
- The entropy of subset S_i can be calculated by means of the formula as shown in the next slice
- P_i^+ is the probability that a randomly taken example in S_i is ⊕ and n_i^+ is the number of ⊕ in S_i $\quad P_i^+ = \frac{n_i^+}{n_i^+ + n_i^-}$
- P_i^- is the probability that a randomly taken example in S_i is ⊗ and n_i^- is the number of ⊗ in S_i $\quad P_i^- = \frac{n_i^-}{n_i^+ + n_i^-}$

Entropy formula

$$H(S_i) = -P_i^+ \log P_i^+ - P_i^- \log P_i^-$$

$$P_i^+ = \frac{n_i^+}{n_i^+ + n_i^-}, P_i^- = \frac{n_i^-}{n_i^+ + n_i^-}$$

Attribute consideration

- Let the values of attributes *at* split the set S of examples into the subset S_i, i=1,…k, then the entropy of the system of subsets S_i is :

$$\mathrm{H(S,at)} = \sum_{i=1}^{k} P(S_i) \bullet H(S_i)$$

- $H(S_i)$ is the entropy of the subset S_i; $P(S_i)$ is the probability of an example belonging to S_i and can be estimated by the relative size of the subset S_i in S:

$$P(S_i) = \frac{|S_i|}{S}$$

Information gain

- Information gain achieved by the partitioning along *at* is measured by the entailed decrease in entropy: I(S, *at*)=H(S) - H(S,*at*)
where H(S) is the a priori entropy of S (before splitting) and H(S,*at*) is the entropy of the system of subsets generated by the value of *at*

Positive and negative examples for concept learning

example	at1	at2	at3	Classification
e1	a	x	n	⊕ (positive)
e2	b	x	n	⊕
e3	a	y	n	⊗ (negative)
e4	a	z	n	⊗
e5	a	y	m	⊕
e6	b	y	n	⊗
e7	b	y	m	⊕
e8	a	x	m	⊕

Computation result

- H(S) before splitting the attribute consideration 5 positive and 3 negative among 8 examples in S, the a priori entropy of the system S is:
  $H(S) = -P^{+}\log P^{+} + -P^{-}\log P^{-}$
  $= -(5/8)\log(5/8) – (3/8)\log(3/8) = 0.31025$
- Log2=0.3010, log3=0.4771, log4=0.6021, log5=0.69897, log6=0.77815, log8=0.9031, log(5/8)=log5-log8=0.69897-0.9031=-0.20413 log(3/8)=log3-log8=0.4771-0.9031=0.426

Computation result

- The number of ⊕ is about the same as the number of ⊗
- If the number of ⊕ were much larger than that of ⊗, we should have a high chance of a correct guess of the class by simply assuming that it is always ⊕ ➔ this would correspond to small entropy

Entropy of the different partition

- at1 has S_a and S_b, it can be computed individually
  S_a has 5 items (3 ⊕ and 2 ⊗)
  S_b has 3 items (2 ⊕ and 1 ⊗)

$$H(S_i) = -P_i^+ \log P_i^+ - P_i^- \log P_i^-$$

$$P_i^+ = \frac{n_i^+}{n_i^+ + n_i^-}, P_i^- = \frac{n_i^-}{n_i^+ + n_i^-}$$

$$\text{H(S,at)} = \sum_{i=1}^{k} P(S_i) \bullet H(S_i)$$

- $H(S_a)$=-(3/5)log(3/5)-(2/5)log(2/5)=0.29231
  $H(S_b)$=-(2/3)log(2/3)-(1/3)log(1/3)=0.276
  H(S,at1)=5/8*(0.29231)+3/8*(0.276)=0.28619

Entropy of the different partition

- at3 has S_m and S_n, it can be computed individually
  S_m has 3 items (3 ⊕)
  S_n has 5 items (2 ⊕ and 3 ⊗)
  $H(S_m)$=-1 log(1)- 0 log(0)=0
  $H(S_n)$=-(2/5)log(2/5)-(3/5)log(3/5)=0.29231
  H(S,at3)=3/8*(0)+5/8*(0.29234)=0.1826937

Entropy of the different partition

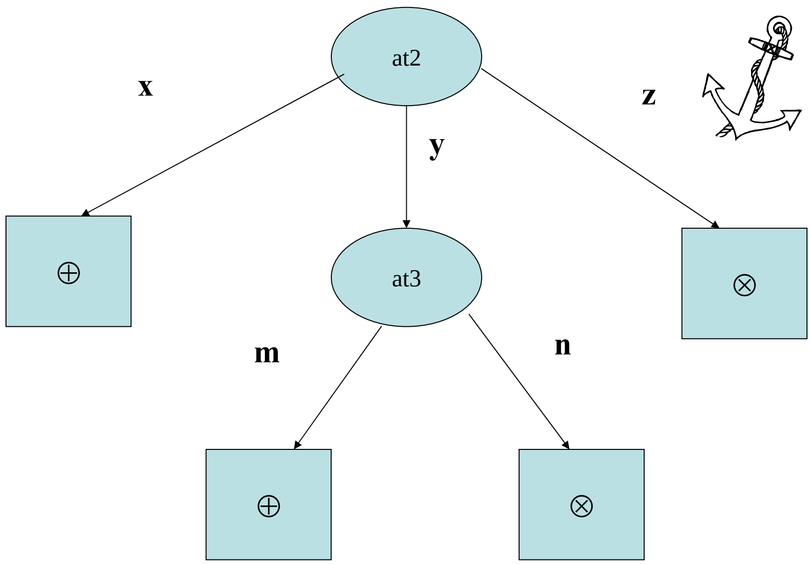
- at2 has S_x ,S_y and S_z, it can be computed individually

  S_x has 3 items (3 ⊕)
  S_y has 4 items (2 ⊕ and 2 ⊗)
  S_z has 1 items (1 ⊗)
  $H(S_x)=-(1)\log(1)-(0)\log(0)=0$
  $H(S_y)=-(2/4)\log(2/4)-(2/4)\log(2/4)=0.3010$
  $H(S_z)=-(0)\log(0)-(1)\log(1)=0$
  $H(S,at2)=3/8*(0)+1/8*(0)+4/8*(0.3010)=0.1505$

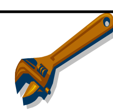
Information gain comparison

- I(S, at1)=H(S)-H(S,at1)
  =0.31025- 0.28619 =0.02406
- I(S, at2)=H(S)-H(S,at2)
  =0.31025- 0.1505=0.15975
- I(S, at3)=H(S)-H(S,at3)
  =0.31025- 0.1826937 =0.1275563
- at2 has the highest information gain, and should be selected as the root of the tree
- Use of entropy is just one of many possibilities

Decision tree of example table



Example for homework



example	at1	at2	at3	Classification
e1	y	n	r	⊕
e2	x	m	r	⊕
e3	y	n	s	⊕
e4	x	n	r	⊕
f1	x	m	s	⊗
f2	y	m	t	⊗
f3	y	n	t	⊗
f4	z	n	t	⊗
f5	z	n	r	⊗
f6	x	n	s	⊗

計算複雜程度

- 本班考試成績的平均值，是針對本班所有同學的考試成績而言。所以“平均值”是指的是廣義集合內各個個體的標誌值的平均值。
- 平均值=（每個個體的標誌值的合計值）/ 個體總數

$$\rightarrow \overline{X} = \sum_i x_i / N$$

- 如果廣義集合的個體總數（*N*）多於100個，用這種公式求平均值容易出現資料登錄差錯。統計學裏還有其他的計算公式就可以減少錯誤

$$\rightarrow \overline{X} = (\sum_i n_i x_i) / N$$

計算複雜程度

- 60個同學的考試分數(分佈函數值) 為100分有20人，90分有30人，80分有10人，平均值=（20×100+30×90+10×80）/ 60 =91.67
- 如果用每個同學的分數相加要做60次加法，而這裏僅做了7次計算就得到出了平均值為91.67，這顯然比前一個公式簡單

計算複雜程度

- 平均值=（∑有該標誌值的個體的數量×該標誌值）/ 個體總數
- 現在把求平均值公式做一些變形就得到了廣義集合的另外一個特徵量 --- 複雜程度。
- 變形 ➔ 平均值的公式裏不是有該標誌值嗎？把它改成“具有該標誌值的個體在總體中占的百分比的對數”
- 平均值公式裏最後不是要用廣義集合內的個體總數N來除嗎？現在省去這個計算，但是在公式最前面加個負號

計算複雜程度

- 複雜程度=-∑每個標誌值的分佈函數值×具有該標誌值的個體在總體中占的百分比的對數
- 如果用C 表示複雜程度、用n_i表示各個標誌值具有的個體的數量、用N 表示廣義集合內個體的總數，複雜程度的公式可以寫為 $C = \sum_i n_i \log(\frac{n_i}{N})$
- 仍以學生考試成績為例，學生成績的複雜程度

=-[20*log(20/60)+30*log(30/60)+10*log(10/60)] = 26.35

- 26.35這個數字描述了60個同學考試成績的複雜程度
- 如果全班同學都是清一色的100分。公式計算的結果是學生考試成績的複雜程度=-60×log（60/60) = 0

複雜程度的單位：bits

- 計算複雜程度的公式中要用“對數”，而對數的值與求對數用什麼為“底”有關。這引導我們從規定“對數的底”是什麼的角度去確定複雜程度的單位。如果規定計算複雜程度時對數都“**以2為底**”，其複雜程度的單位就稱為“Bit”
- “Bit”是描述資訊存儲量大小的單位。為什麼複雜程度要用電腦界的單位？電腦界借用了表示資訊的單位（Bit）而複雜程度的計算公式與資訊量的計算公式是一致的，所以我們也借用資訊理論中對資訊的計量單位。

複雜程度的單位：bits

- 如果計算時用的是以10位底的對數，也可以用對數換底公式換算成Bit，把以10為底的對數值乘以3.3219就得到了以2為底的對數值(比特值)。即$\log_2 x = 3.321928*\log_{10} x$
- 有個例子是擲了1000次的硬幣，正反面朝上的事件各為500次。由這個結局組成的廣義集合的複雜程度301.0 (=-[500×log(500/1000)+[500×log(500/1000)] =-1000×log(1/2)=301.0)，把它乘3.321928就得到了以Bit單位的複雜程度，即301×3.321928=1000Bit。

複雜程度的單位：bits

- 對數的計算現在都用電腦或者計數器完成。而對數都是10位底或者以自然數e（2.71828）為底。所以在一些場合這都要利用對數換底公式去換算成“Bit”。
- 如果直接用以10為底計算對數，得到的複雜程度應當稱為Hartley，以自然數e為底計算對數而得到的複雜程度應當稱為nat。這些單位也是從資訊理論中借來的，但是它們都沒有“Bit”那麼常用。

Computation result

- H(S) before splitting the attribute consideration 5 positive and 3 negative among 8 examples in S, the a priori entropy of the system S is:
  $H(S) = -P^{+}\log P^{+} + -P^{-}\log P^{-}$
  $= -(5/8)\log(5/8) – (3/8)\log(3/8) = 0.31025$
- $\log_2 x = 3.321928 * \log_{10} x$
- H(S)= 0.31025 * 3.321928 = 0.954 bits

Entropy of the different partition

- at1 has S_a and S_b, it can be computed individually

  S_a has 5 items (3 ⊕ and 2 ⊗)

  S_b has 3 items (2 ⊕ and 1 ⊗)

  $H(S_a)$=-(3/5)log(3/5)-(2/5)log(2/5)=0.29231

  $H(S_b)$=-(2/3)log(2/3)-(1/3)log(1/3)=0.276

  H(S,at1)=5/8*(0.29231)+3/8*(0.276)=0.28619 = 0.28619 * 3.321928 = 0.951bits

Entropy of the different partition

- at3 has S_m and S_n, it can be computed individually

  S_m has 3 items (3 ⊕)

  S_n has 5 items (2 ⊕ and 3 ⊗)

  $H(S_m)$=-1 log(1)- 0 log(0)=0

  $H(S_n)$=-(2/5)log(2/5)-(3/5)log(3/5)=0.29231

  H(S,at3)=3/8*(0)+5/8*(0.29234)=0.1826937 = 0.1826937 * 3.321928 = 0.6069bits

Entropy of the different partition

- at2 has S_x ,S_y and S_z, it can be computed individually

  S_x has 3 items (3 ⊕)
  S_y has 4 items (2 ⊕ and 2 ⊗)
  S_z has 1 items (1 ⊗)
  $H(S_x)=-(1)\log(1)-(0)\log(0)=0$
  $H(S_y)=-(2/4)\log(2/4)-(2/4)\log(2/4)=0.3010$
  $H(S_z)=-(0)\log(0)-(1)\log(1)=0$

H(S,at2)=3/8*(0)+1/8*(0)+4/8*(0.3010)=0.1505
=0.1505 * 3.321928 = 0.5bits

Information gain comparison

- I(S, at1)=H(S)-H(S,at1)
  = 0.954 bits – 0.951bits =0.03bits
- I(S, at2)=H(S)-H(S,at2)
  = 0.954 bits – 0.5bits=0.454 bits
- I(S, at3)=H(S)-H(S,at3)
  = 0.954 bits - 0.6069bits =0.347bits
- at2 has the highest information gain, and should be selected as the root of the tree
- Use of entropy is just one of many possibilities

Pruning the trees

- A few pitfalls can put the use of a decision tree in question ➔ over-fitting
- A tree branch (ending with a class label) might have been created from examples that are noisy ➔the attribute values or class labels are erroneous
- This branch or some of its decision tests will be misleading
- If the number of attributes is large, the tree may contain tests on random features that are irrelevant for correct classifications

Pruning the tree

- Very large trees are hard to interpret and the user will perceive them as a black box representation
- It may be beneficial to prune the resulting tree
- 2 approaches to prune the decision tree
  - On-line pruning: stop the tree growing when the information gain caused by the partitioning of the example set falls below a certain threshold
  - Post-pruning: prune out some of the branches after the tree has been completed

Tree simplification (On-line pruning)

- Minimal-error pruning aims at pruning the tree to such an extent that the overall expected classification error on new examples is minimized → the classification error is estimated for each node in the tree
- In the leaves, the error is estimated using one of the methods for estimating the probability that a new object falling into this leaf will be misclassified

Tree simplification (On-line pruning)

- Suppose that N is the number of examples that end up in the leaf, and e is the number of these examples that are misclassified at this leaf. The Laplace estimate (e+1)/(N+k) (where k is the number of all the classes) is used to estimate the expected error

Tree simplification (On-line pruning)

- For a non-leaf node of the decision tree, its classification error is estimated as the weighted sum of the classification errors of the node's subtree. The weights are calculated as relative frequencies of examples passing from the node into the corresponding sub-trees
- The non-leaf error estimate is called a back-up error (threshold!!)

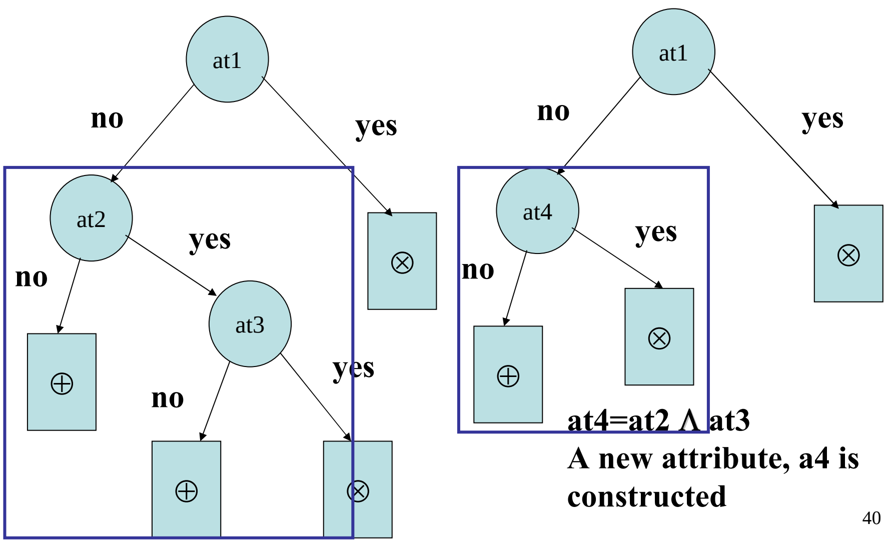
Tree simplification (On-line pruning)

- If the error estimate is lower than the backup error, the subtrees will be pruned out
- The process of pruning subtrees starts at the bottom levels of the tree and propagating upwards as long as the backed-up errors are higher than the 'static estimates'

Tree simplification

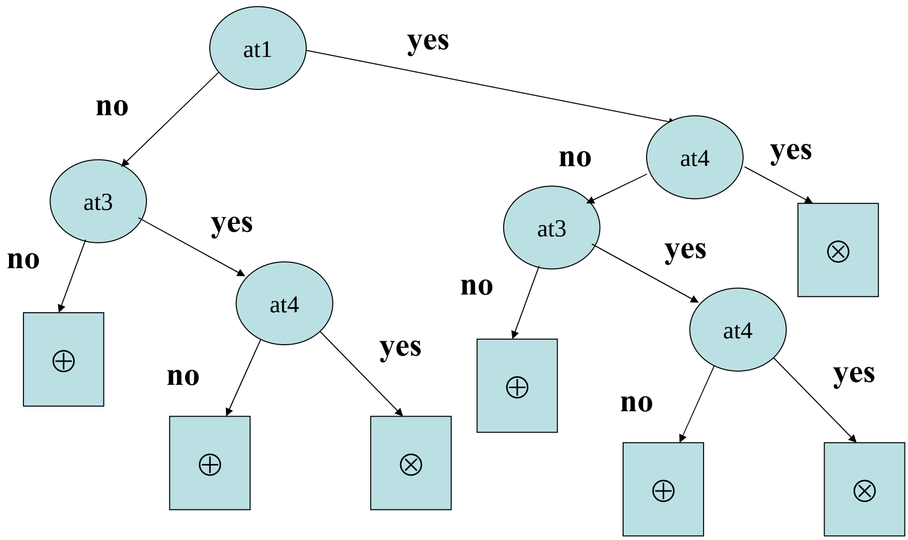
- Another type of tree pruning by carrying out a kind of constructive induction
- Learning system strives to create new attributes as logical expressions over the attributes provided by the teacher
- Constructive induction can be profitable where a sub-tree is replicated in more than one position in the tree

Constructive induction in decision tree

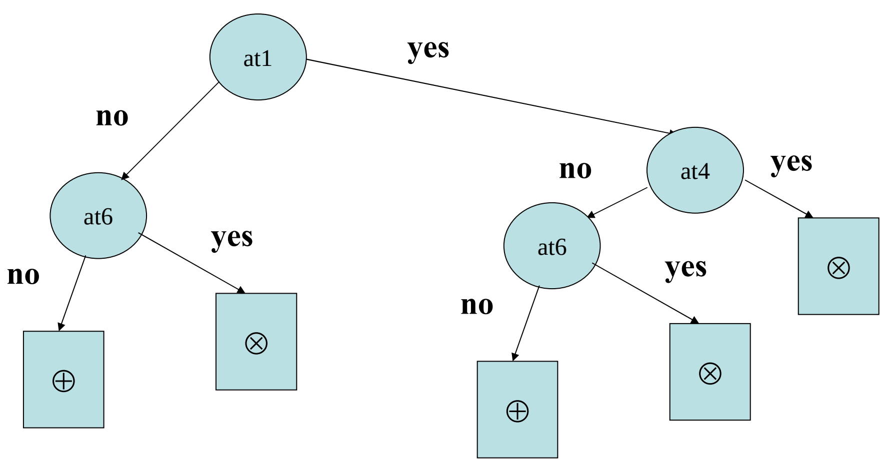


at4=at2 Λ at3
A new attribute, a4 is constructed

The replication problem in decision tree



Simplified version of a decision tree

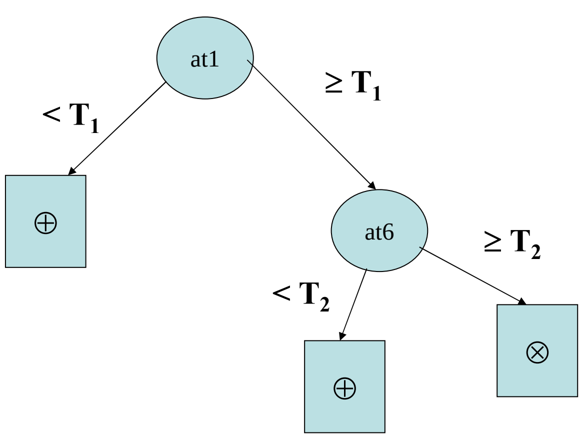


Threshold position

- Decision trees can also be induced from numerical attributes not only symbolic attributes
- On possible method is to provide one additional step, the binarization of the numerical attributes
- At each node, the respective attribute value is tested against threshold T_i
- Threshold position in the range of values can be determined by entropy
- First order all the examples according to the best attribute and observe the classification values

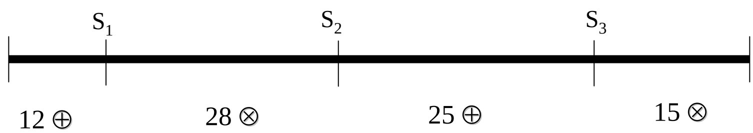
Cope with numeric data

- Thresholds the numerical ranges into pairs of subintervals to be treated as symbols
- Decision tree built from numeric data as shown



Threshold position

- The classification value of ⊕ and ⊗ decompose the set of 80 examples into 4 regions
- The candidate splitting cuts lie on the boundaries between the regions ➔ the cut with the highest information gain is selected



Numeric version of TDIDT Algorithm

1. Use the entropy measure to find the optimal split of the numeric attributes
2. Determine the attribute whose optimal split maximizes entropy and partition the example set along this attribute into 2 subsets
3. If the termination criterion is not satisfied, repeat the procedure recursively for each subset
4. With each new sub-tree, the splitting cuts must be recalculated