Chapter 3

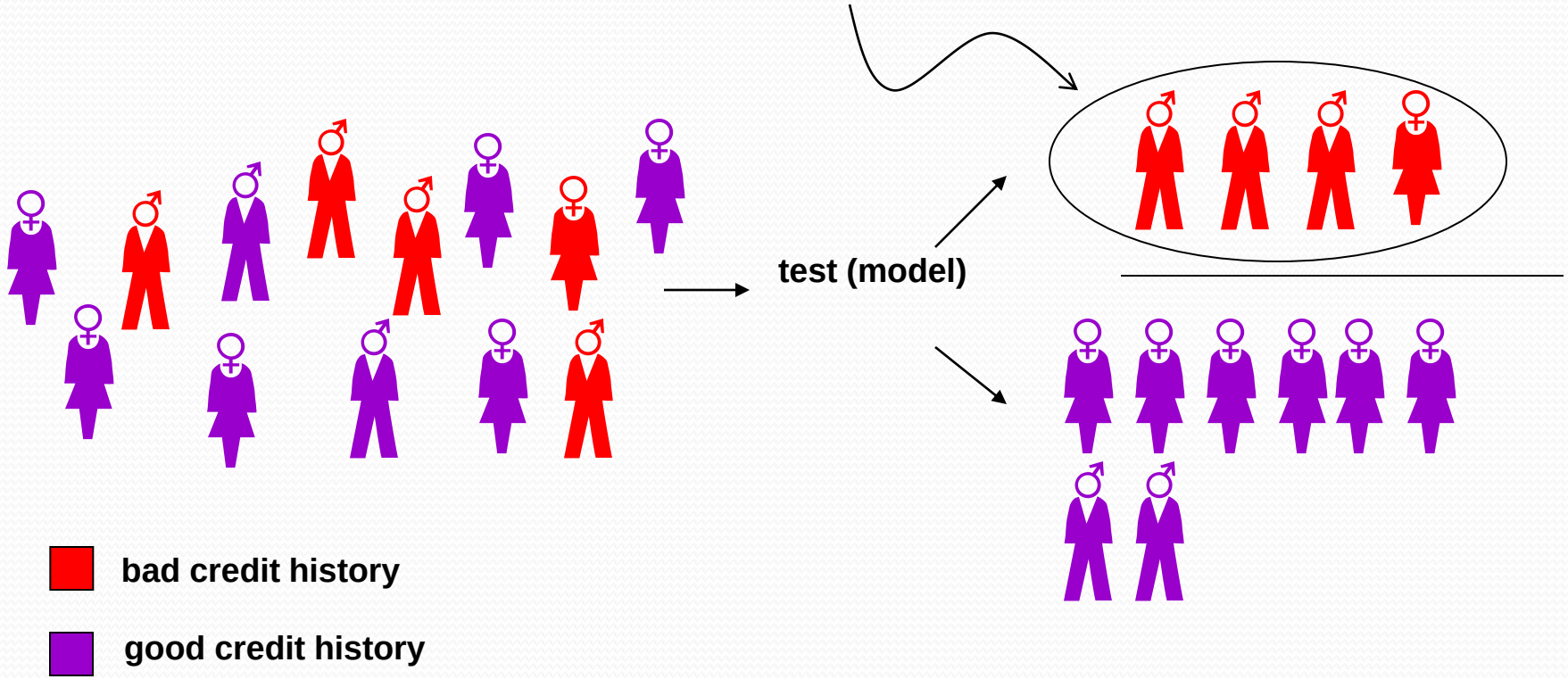
Classification (Part 1)

Overview

- Basic decision tree classifier (DT) construction
- Some technical issues of DT classification
- Evaluating classifiers

Classification

What common characteristics are shared by the red people, and not by the purple people?



Independent attributes

class

Record id	Age	Income	Student	Credit-rating	Own-computer
1	< 30	High	No	Bad	No
2	< 30	High	No	Good	No
3	30 .. 40	High	No	Bad	Yes
4	> 40	Medium	No	Bad	Yes
5	> 40	Low	Yes	Bad	Yes
6	> 40	Low	Yes	Good	No
7	30 .. 40	Low	Yes	Good	Yes
8	< 30	Medium	No	Bad	No
9	< 30	Low	Yes	Bad	Yes
10	> 40	Medium	Yes	Bad	Yes
11	< 30	Medium	Yes	Good	Yes
12	30 .. 40	Medium	No	Good	Yes
13	30 .. 40	High	Yes	Bad	Yes
14	> 40	Medium	No	Good	No

Example rules

- If age < 30 and is not a student ⇒ *not a computer owner*
- If age < 30 and is a student ⇒ *a computer owner*
- If age is between 30 to 40 ⇒ *a computer owner*
- If age > 40 with a good credit rating ⇒ *not a computer owner*
- If age > 40 with a bad credit rating ⇒ *a computer owner*

Data Model

- A dataset consists of a number of records.
- Each record consists of a number of attribute values.
- One particular (nominal) attribute is called the *label* (or *class* or *dependent attribute*).
- Records that share the same label value form a *class*.
- We want to discover rules that help predict, given a future (unclassified) record, which class the record belongs.

Supervised Learning

- Our general approach to the classification problem:
  - prepare a dataset of labeled records
  - draw a random sample (e.g., 80%), call it the *training set*
  - use the training set to train a classifier
  - apply the classifier to the rest (e.g., 20%) of the records (the *test set*) to evaluate the accuracy of the classifier
  - during the training phase, the classifier is fed with labeled records (*examples of each class*), the learning is supervised. This machine learning approach is called *supervised learning*.
  - For the purpose of parameter tuning, we may divide the data into *training-validation-test* sets.

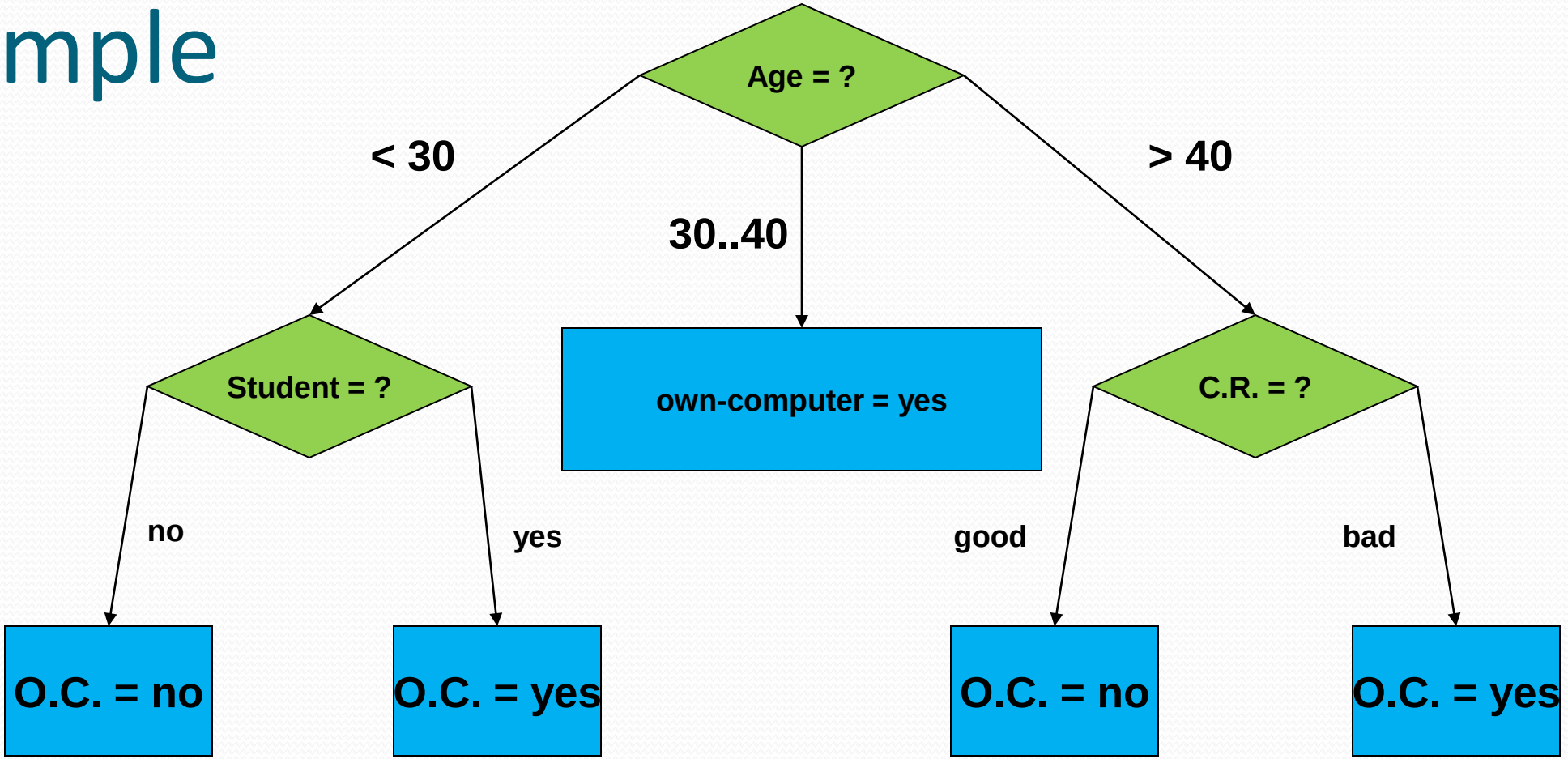
Decision-Tree Classifier (DT)

- We consider how to build a *decision-tree classifier*
- We assume that attributes are all *nominal* – they only take on a finite set of *named values*
- Numeric attributes can be mapped to nominal attributes by *discretization* or *binning*
- Example:
  - income can be grouped into
    - low (< 80K), medium (>=80K, <250K), high (>=250K)
  - a person's height can be
    - very short, short, average, tall, very tall (with a suitable mapping)

Decision-Tree Classifier

- A decision tree is a tree structure
- Properties:
  - each *internal node* denotes a *test* on an attribute
  - each *branch* from a node represents an *outcome* of a test
  - each *leaf node* is associated with a *class label*

Example


Age = ?
< 30
30..40
> 40
Student = ?
own-computer = yes
C.R. = ?
no
yes
good
bad
O.C. = no
O.C. = yes
O.C. = no
O.C. = yes

Entropy

- (Shannon) Entropy is a quantitative measure of *uncertainty* or *randomness*
- Consider an information source that generates symbols (‘*a*’ or ‘*b*’)

Entropy

- Q: How certain is the recipient in *predicting* the next symbol the information source would generate?
- A: Depends on the *probabilities* with which the source generates the symbols

Entropy

- Examples:
  - if the source sends ‘a’ and ‘b’ with equal probability, then the recipient is *very uncertain*
  - if the source always sends ‘a’ and not ‘b’, then the recipient is *very certain*
  - if the source sends ‘a’ with a probability of 0.9, and sends ‘b’ with a probability of 0.1, then the recipient is *pretty certain*
  - what if the source could send 3 symbols ‘a’, ‘b’, and ‘c’ with probabilities 0.3, 0.6, and 0.1, respectively? How certain is the recipient?

Entropy

- Entropy is a *numerical measure* that quantifies the concept of *uncertainty.*
- According to Shannon, the entropy of an information source, S, is defined as:

$$H(S) = \sum_i p_i \log_2 \frac{1}{p_i}$$

where p_i is the probability that the i-th symbol occurs.

- Larger entropy $\Leftrightarrow$ higher uncertainty

Example

- if the information source sends:
  - ‘a’ (0.5), ‘b’ (0.5)
    - entropy = $0.5 * \log_2 (1/0.5) + 0.5 * \log_2 (1/0.5) = 1$
  - ‘a’ (1.0), ‘b’ (0.0)
    - entropy = $1.0 * \log_2 (1/1.0) = 0$
  - ‘a’ (0.9), ‘b’ (0.1)
    - entropy = $0.9 * \log_2 (1/0.9) + 0.1 * \log_2 (1/0.1) = 0.469$
  - ‘a’ (0.3), ‘b’ (0.6), ‘c’ (0.1)
    - entropy = $0.3 * \log_2 (1/0.3) + 0.6 * \log_2 (1/0.6) + 0.1 * \log_2 (1/0.1) = 1.295$

Very uncertain

Very certain

A bit uncertain

Very very uncertain

Entropy

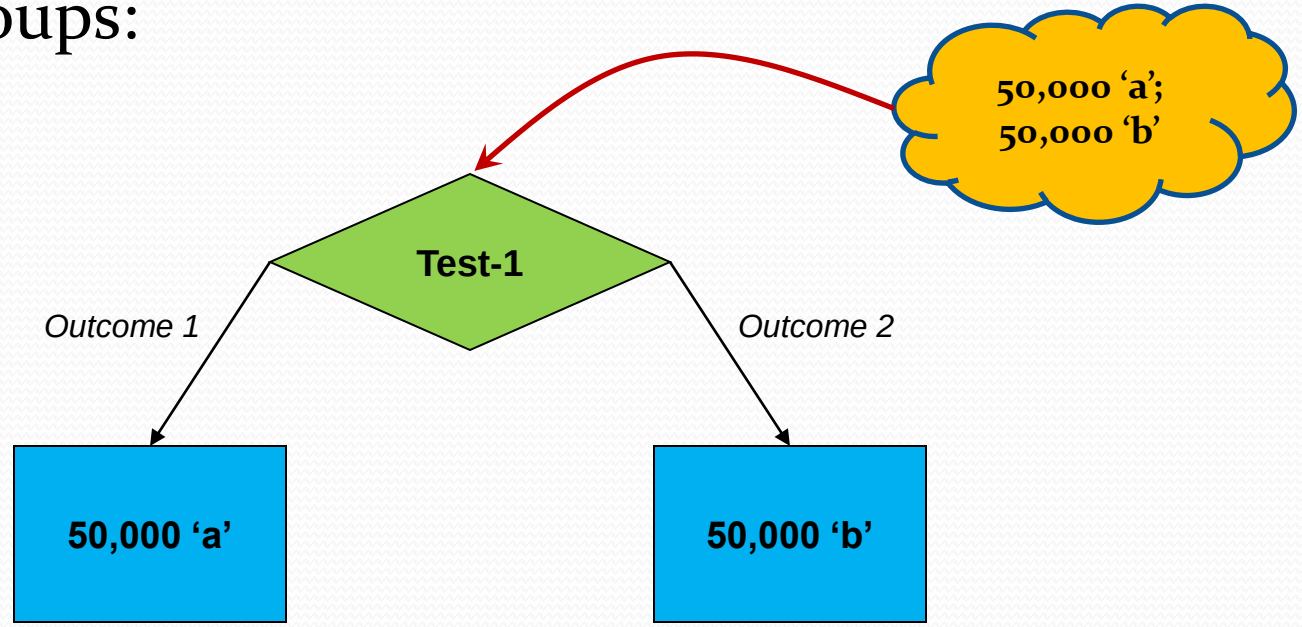
- The concept of entropy is used to select the tests used in a decision tree.
- Consider a dataset of 100,000 records with
  - 50,000 labeled ‘a’
  - 50,000 labeled ‘b’
  - If you are given a new record (the 100,001-st) and you are asked to give it a label, how uncertain are you?
  - Using entropy, your amount of uncertainty (without any additional information) is 1.0

Entropy

- consider a dataset of 100,000 records with
  - 90,000 labeled ‘a’
  - 10,000 labeled ‘b’
  - If you are given a new record (the 100,001st) and you are asked to give it a label, how uncertain are you?
  - Using entropy, your amount of uncertainty (without any additional information) is only 0.469. In fact, you are somewhat sure that the record should be labeled ‘a’.

Finding a Good Test

- Consider the case with 50,000 ‘a’ records and 50,000 ‘b’ records again
- Suppose you come up with a test (Test-1) that partitions the 100,000 records into two groups:

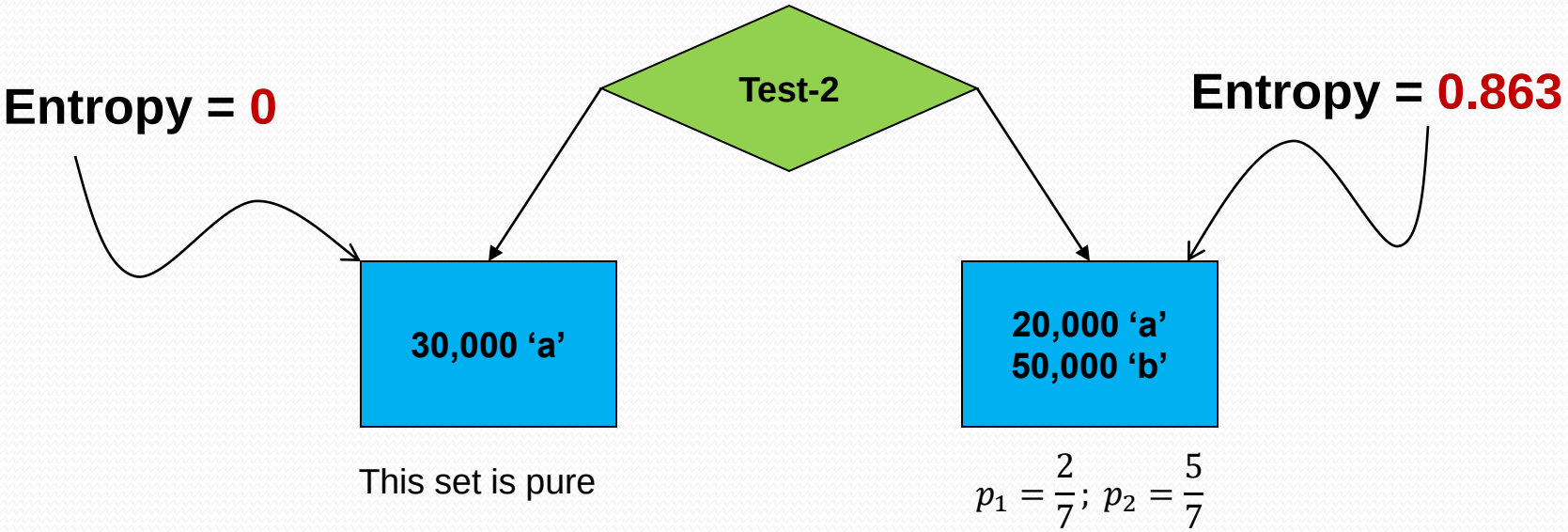


Finding a Good Test

- How good is Test-1?
  - *it is perfect!*
- Test-1 successfully classified all 100,000 records.
- Given an unlabeled record *x*, we can apply Test-1 on *x*. If Test-1 indicates that *x* should go to the
  - left group: we are very certain that *x* should be labeled 'a' because *x* is sharing the group with 50,000 'a' records and not with any 'b' record
  - right group: *x* should be labeled 'b'
- Note that the entropy of either group is 0

Finding a Good Test

- consider another test Test-2

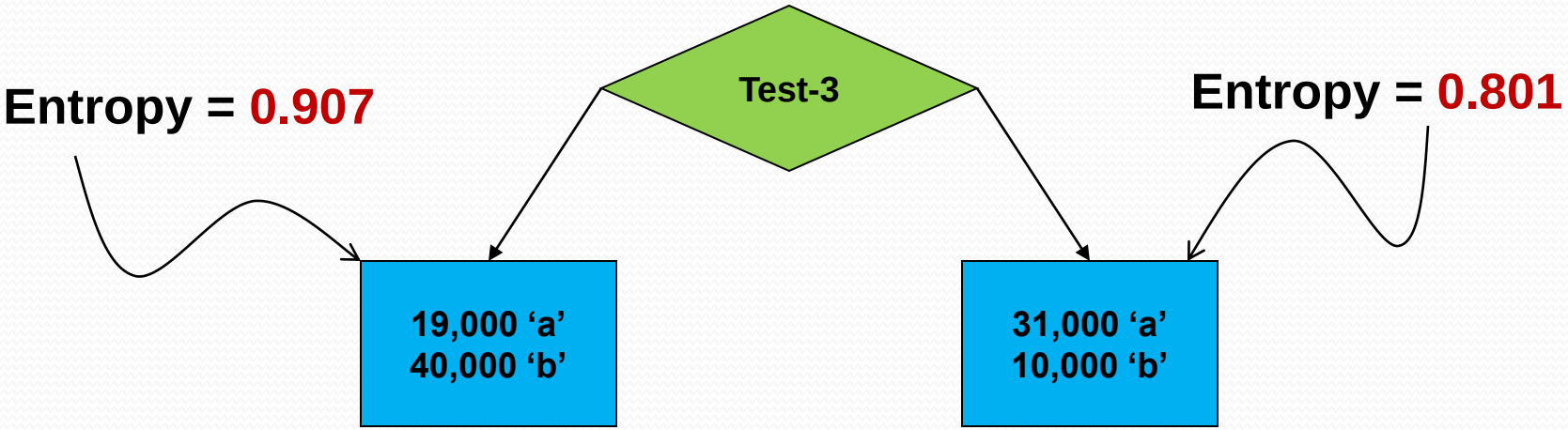


Finding a Good Test

- Since 30% of the records are put to the left group by Test-2 (and 70% to the right), given a new record, *x*, we would expect that *x* go to the left group (after Test-2) with a 30% probability
- The expected entropy (after Test-2) is thus:
  - 0.3 * 0 + 0.7 * 0.863 = 0.604
- Test-2 is a *good* test in the sense that it reduces the entropy from 1.0 to 0.604.
- 1-0.604 = 0.396 is called the *information gain* of the test.

Finding a Good Test

- consider yet another test Test-3



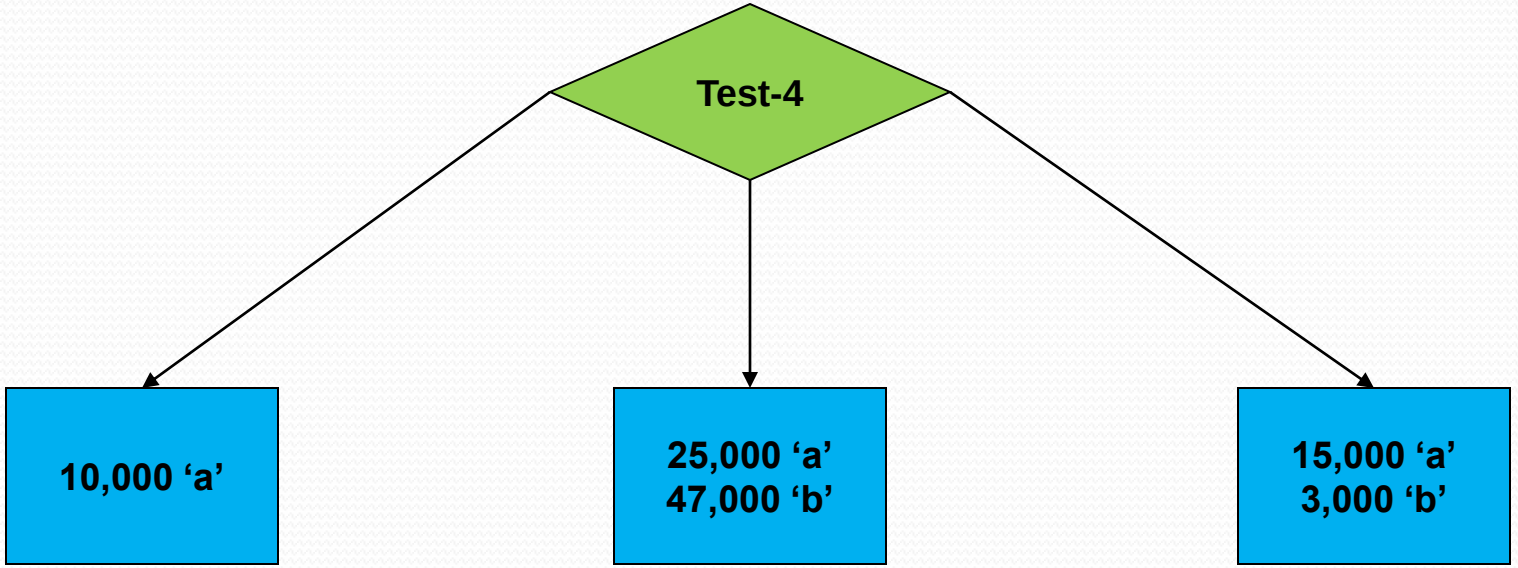
- Expected entropy after Test-3:
  - (59/100) * 0.907 + (41/100) * 0.801 = 0.864

Finding a Good Test

- Ranking the 3 tests:
  - Test-1 is the best (highest information gain)
  - Test-2 is the second best
  - Test-3 is the worst

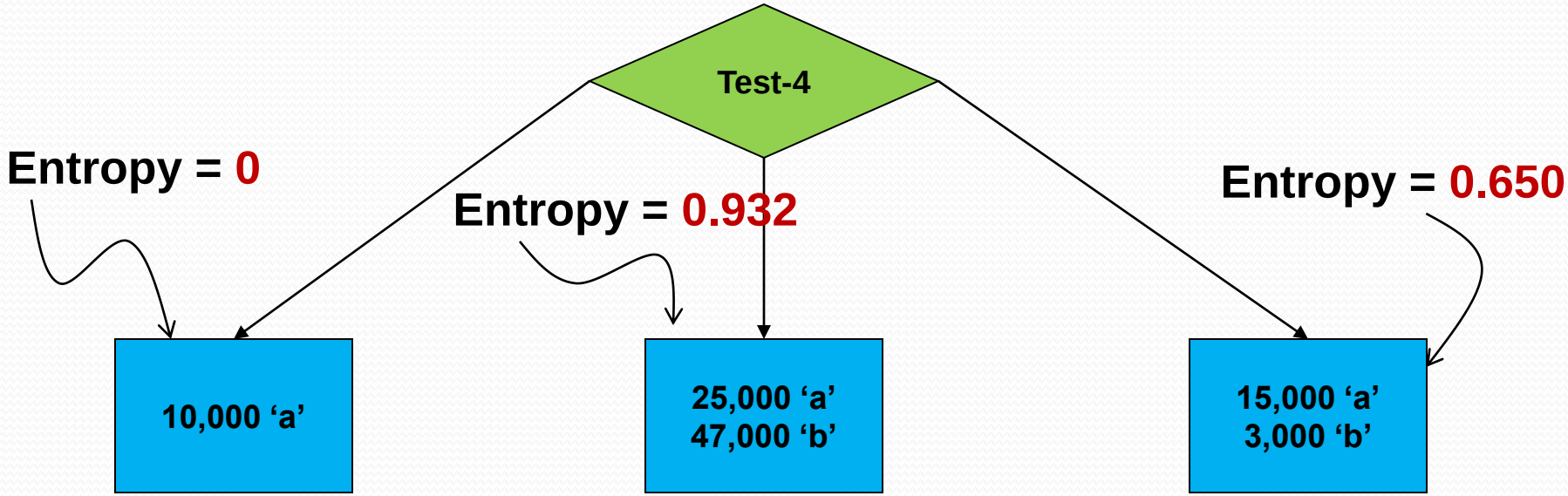
Finding a Good Test

- What about this test?



Finding a Good Test

- What about this test?



expected entropy = 0.1 * 0 + 0.72 * 0.932 + 0.18 * 0.65 = 0.788

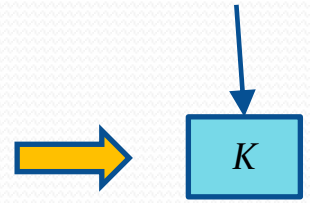
Build a Decision Tree

- Based on the concept of test and entropy
- Call our algorithm build-tree(D,L), where
  - D is a training set of data records whose class labels are known
  - L is a list of nominal independent attributes
- Goal: build a decision tree using tests on attributes' values
- A recursive construction algorithm

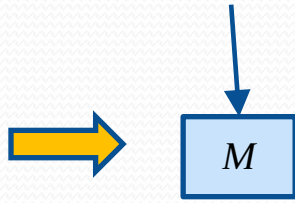
Procedure (base cases)

- If …
  - all records in D are of the same label K (i.e., D is pure)
    - build-tree(D,L) returns a tree with only a leaf node labeled K. No test is needed
  - L is empty ⇒ no tests can be derived
    - build-tree(D,L) returns a tree with only a leaf node labeled M, where M = most common label among the records in D.
  - all records in D share the same attribute values for all attributes in L ⇒ no meaningful tests can be derived
    - Same action as this one.

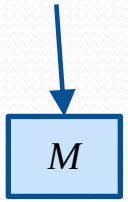
a	b	c	**K**
a	d	e	**K**
x	b	c	**K**
x	b	y	**K**



M
X
M
M

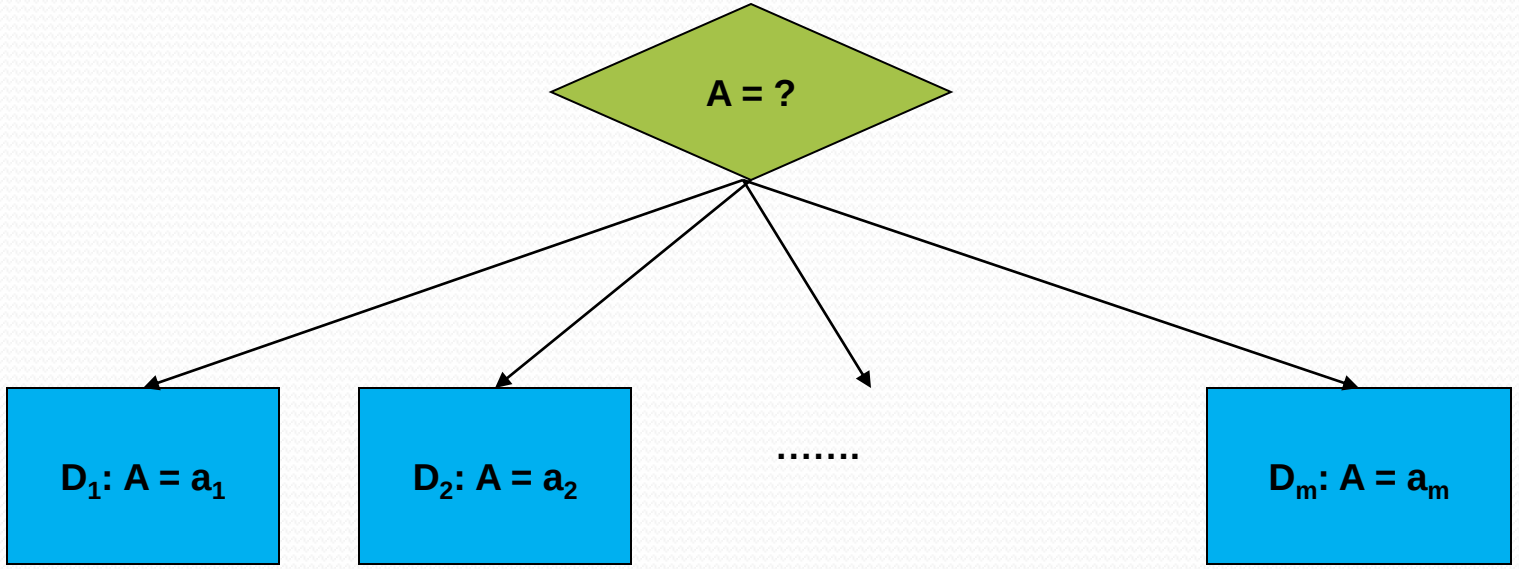


a	b	c	**M**
a	b	c	**X**
a	b	c	**M**
a	b	c	**M**



Procedure (recursive case)

- Otherwise, compute the most effective (i.e., most entropy reducing) test that can be derived from the attributes in *L*. Let the test be



Procedure

- Suppose the test divides D into m groups: D_1 … D_m
- Create a node labeled "A=?"
- For each group, D_i,
  - if D_i is empty, create a subtree T_i with a lone leaf node labeled M, where M is the most common label among the records in D
  - else
    - recursively call build-tree(D_i, L-{A}) to build a subtree T_i,
    - connect the test node "A=?" to T_i by a branch labeled "a_i" (the i-th outcome of the test).

Record id	Age	Income	Student	Credit-rating	Own-computer
1	< 30	High	No	Bad	No
2	< 30	High	No	Good	No
3	30 .. 40	High	No	Bad	Yes
4	> 40	Medium	No	Bad	Yes
5	>40	Low	Yes	Bad	Yes
6	> 40	Low	Yes	Good	No
7	30 .. 40	Low	Yes	Good	Yes
8	< 30	Medium	No	Bad	No
9	< 30	Low	Yes	Bad	Yes
10	> 40	Medium	Yes	Bad	Yes
11	< 30	Medium	Yes	Good	Yes
12	30 .. 40	Medium	No	Good	Yes
13	30 .. 40	High	Yes	Bad	Yes
14	> 40	Medium	No	Good	No

Example

- Dataset *D*:
  - 14 records
  - label attribute: "Own-computer" (O.C.)
  - 9 computer owners, 5 non-owners
  - entropy of *D*
    - $5/14 * \log_2 (14/5) + 9/14 * \log_2 (14/9) = 0.940$
  - 4 nominal attributes for deriving tests
    - *L* = {'Age', 'Income', 'Student', 'Credit-rating'}
  - goal: to derive rules that would classify a person as a computer owner or not

Example

- Determine the effectiveness of 4 tests
- The test: "Age = ?", e.g., divides *D* into 3 groups:

Group 1: Age < 30

Record id	Age	Income	Student	Credit-rating	Own-computer
1	< 30	High	No	Bad	No
2	< 30	High	No	Good	No
8	< 30	Medium	No	Bad	No
9	< 30	Low	Yes	Bad	Yes
11	< 30	Medium	Yes	Good	Yes

Group 2: Age: 30..40

Record id	Age	Income	Student	Credit-rating	Own-computer
3	30 .. 40	High	No	Bad	Yes
7	30 .. 40	Low	Yes	Good	Yes
12	30 .. 40	Medium	No	Good	Yes
13	30 .. 40	High	Yes	Bad	Yes

Group 3: Age > 40

Record id	Age	Income	Student	Credit-rating	Own-computer
4	> 40	Medium	No	Bad	Yes
5	>40	Low	Yes	Bad	Yes
6	> 40	Low	Yes	Good	No
10	> 40	Medium	Yes	Bad	Yes
14	> 40	Medium	No	Good	No

Example

- Entropy of:
  - group 1 = $2/5 * \log_2 (5/2) + 3/5 * \log_2 (5/3) = 0.971$
  - group 2 = 0
  - group 3 = $2/5 * \log_2 (5/2) + 3/5 * \log_2 (5/3) = 0.971$

- Expected entropy (after test using the "Age" attribute) = $(5/14 * 0.971) + (4/14 * 0) + (5/14 * 0.971) =$ 0.694

Example

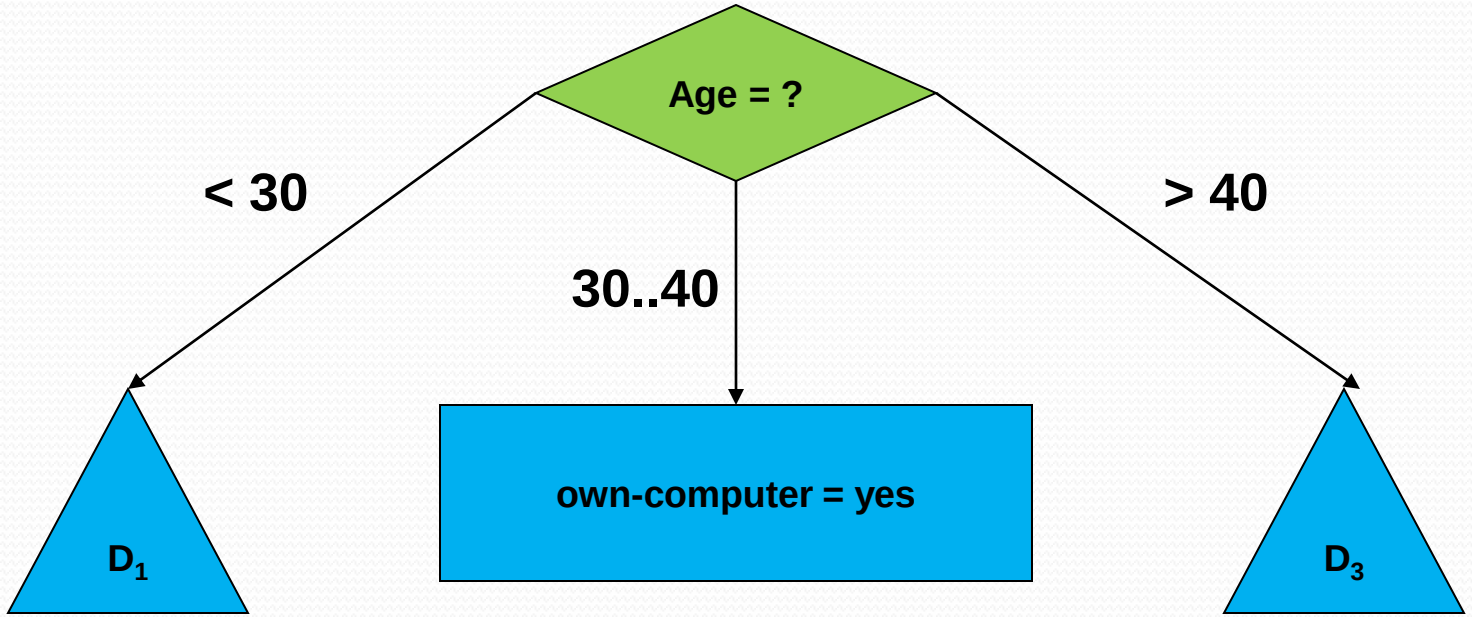
Best

attribute test	expected entropy
Age	0.694
Income	0.911
Student	0.788
Credit-rating	0.892

Example

- "Age" wins out.
- D is divided into 3 groups:
  - D_1 (Age < 30) = {1, 2, 8, 9, 11}
  - D_2 (Age: 30..40) = {3, 7, 12, 13}
  - D_3 (Age > 40) = {4, 5, 6, 10, 14}
- Recursively apply build-tree to the three datasets, with L = {"Income", "Student", "Credit-rating"}
- Since all records in D_2 are of the same label (own-computer = yes), a node is created labeled "own-computer = yes"

A partially-built tree



Example

- For D_1, we have

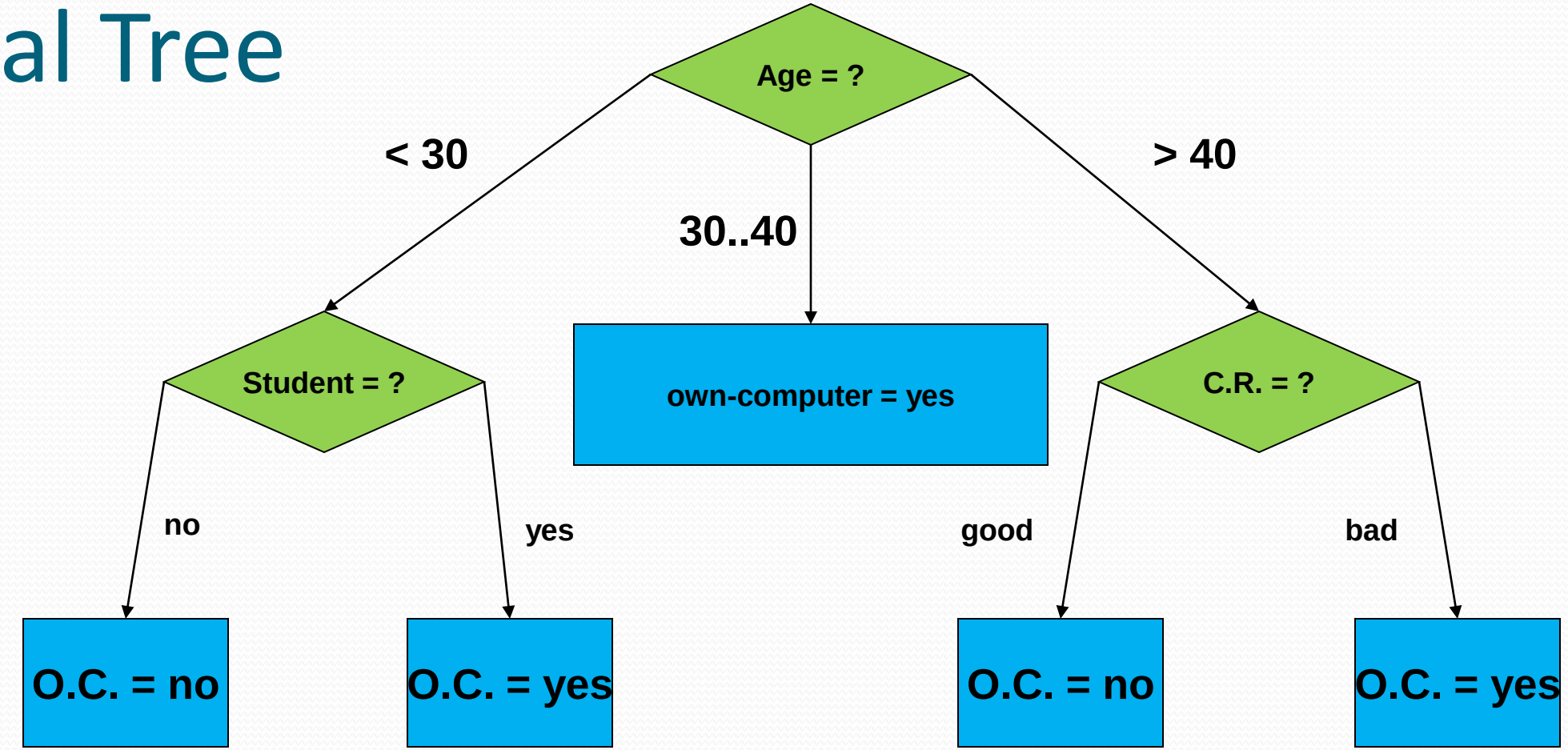
attribute test	expected entropy
Income	0.4
Student	0
Credit-rating	0.951

Example

- For D_3, we have

attribute test	expected entropy
Income	0.951
Student	0.951
Credit-rating	0

Final Tree


Age = ?
< 30
30..40
> 40
Student = ?
own-computer = yes
C.R. = ?
no
yes
good
bad
O.C. = no
O.C. = yes
O.C. = no
O.C. = yes

Rules

- We derive classification rules from the decision tree by *walking* the tree from the root to every leaf.
  - If age < 30 and is not a student ⇒ not a computer owner
  - If age < 30 and is a student ⇒ a computer owner
  - If age is between 30 to 40 ⇒ a computer owner
  - If age > 40 with a good credit rating ⇒ not a computer owner
  - If age > 40 with a bad credit rating ⇒ a computer owner

Applying Rules

- Given a record *X*:

Age	*Income*	*Student*	*Credit-rating*
< 30	medium	yes	fair

is *X* a computer-owner or not?

Algorithm build-tree(D, L) {

Begin

If all records in D share the same label ‘K’

Return a leaf node with the label ‘K’;

If ((L is empty) || (all records in D share the same attribute values)) {

Let ‘K’ be the most common label in D;

Return a leaf node with the label ‘K’;

}

For each attribute A in L do

Compute the expected entropy achieved by using A as the test on D;

Let A be the attribute which achieves the smallest expected entropy;

Create a node N with label “A = ?”;

Assume A has n possible values a_1, …, a_n

Partition D into D_1, …, D_n, where D_i contains all the records with A = a_i;

For each a_i do {

If D_i is empty {

Let ‘K’ be the most common label in D;

Attach a leaf node to N labeled ‘K’;

}

Else

Attach the sub-tree obtained by calling build-tree(D_i, L-{A}) to N;

}

Return the tree rooted at N;

End

Issues

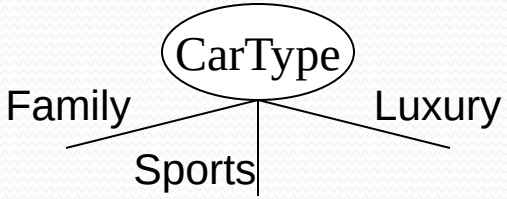
- How to derive a test based on an attribute?
- How to measure the purity (certainty, uncertainty) of a dataset?
- How to measure the performance of a classifier?
- How to deal with numerical attributes?
- Overfitting

How to derive a test?

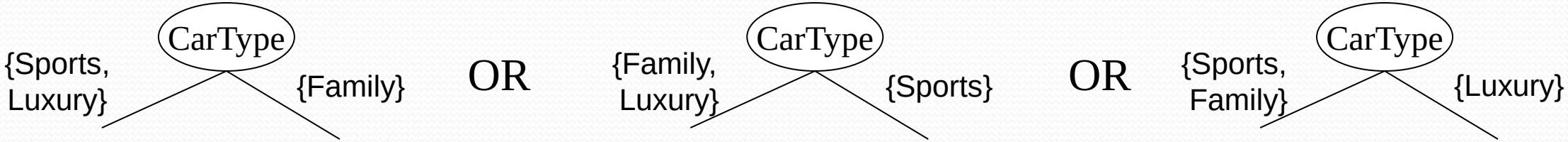
- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous (numerical)
- Depends on number of ways to split (number of outcomes of a test)
  - 2-way split
  - Multi-way split

Splitting (Nominal Attributes)

- *Multi-way split*: Use as many partitions as number of distinct values.

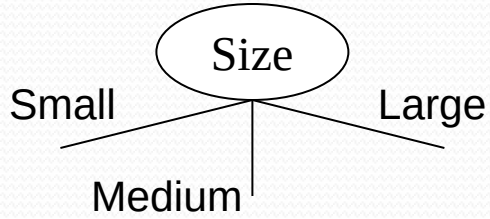


- *Binary split*: Divides values into two subsets.



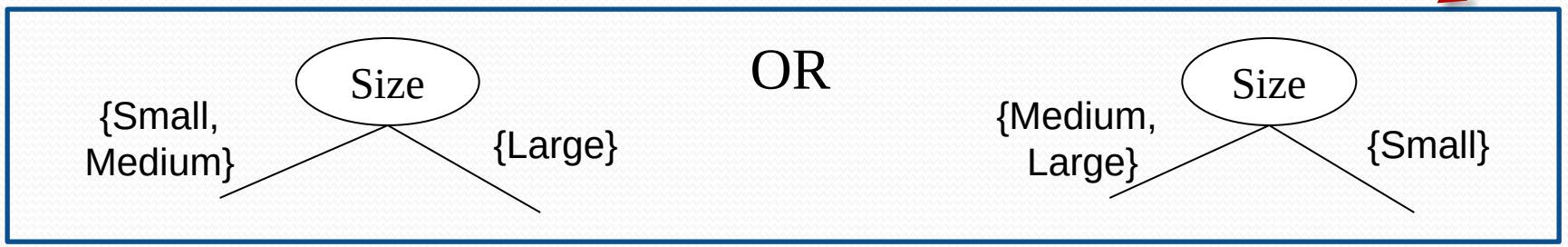
Splitting (Ordinal Attributes)

- *Multi-way split*: Use as many partitions as number of distinct values.



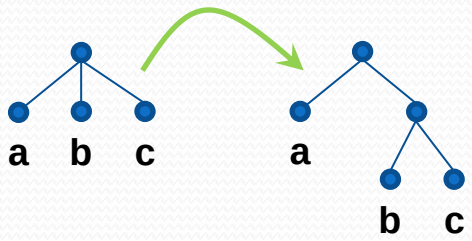
Order preserving

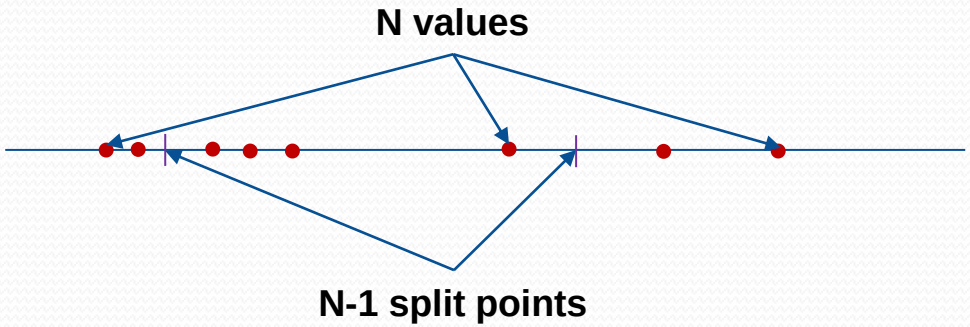
- *Binary split*: Divides values into two subsets.



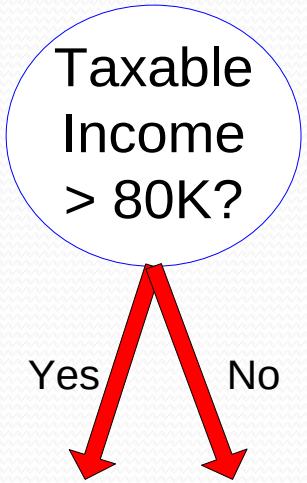
Splitting (Continuous Attributes)

- *Discretization*: transform it to an ordinal attribute
- *Binary Decision*: $(A < v)$ or $(A \geq v)$ — *Split point*
  - consider all possible splits and finds the best cut
  - more computationally expensive
- Multiway split can be achieved by a number of binary splits.

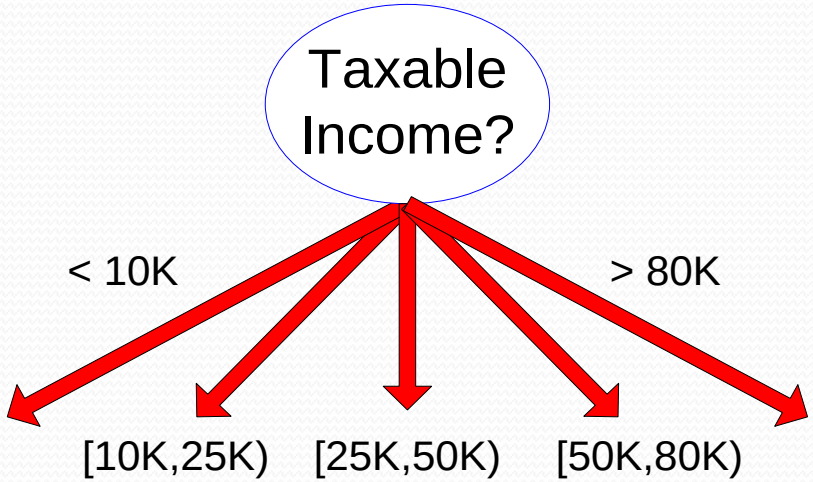




Splitting (Continuous Attributes)



(i) Binary split



(ii) Multi-way split

Impurity measures

- Entropy (Information Gain)
- Gini Index
- Classification error
- Gain Ratio

(different decision-tree algorithms use different purity/impurity measures)

GINI

- Used in CART
- Given a set S:

$$GINI(S) = 1 - \sum_i (p_i)^2$$

  - Maximum $(1 - 1/n_c)$, where n_c is the number of classes
  - Minimum (0.0) when all records belong to one class

Example (GINI)

$$GINI(S) = 1 - \sum_i (p_i)^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)² – (5/6)² = 0.278

C1	**2**
C2	**4**

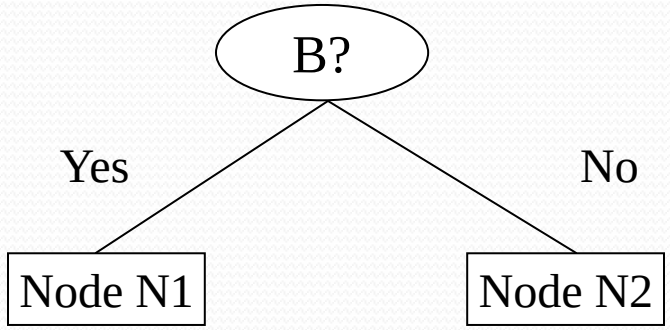
P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)² – (4/6)² = 0.444

Example (GINI)

Before partitioning

	Parent
C1	6
C2	6
Gini = 0.500	



	N1	N2
C1	5	1
C2	2	4
Gini=0.371		

After partitioning

Gini(N1)
= 1 – (5/7)² – (2/7)²
= 0.408

Gini(N2)
= 1 – (1/5)² – (4/5)²
= 0.320

Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.320
= 0.371

Entropy (Information Gain)

- Used in ID3
- Given a set S:

$$Entropy(S) = \sum_i p_i \log_2 (1/p_i)$$

  - Maximum ($\log n_c$), where n_c is the number of classes
  - Minimum (0.0) when all records belong to one class

Example (Entropy/Information Gain)

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 **P(C2) = 6/6 = 1**

Entropy = 0

C1	**1**
C2	**5**

P(C1) = 1/6 **P(C2) = 5/6**

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (5/6) = 0.65

C1	**2**
C2	**4**

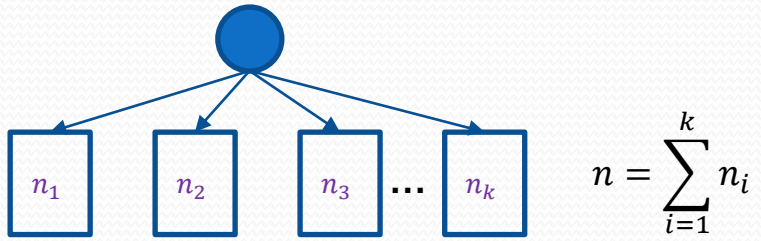
P(C1) = 2/6 **P(C2) = 4/6**

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Entropy/Information gain

- Disadvantage: Tends to prefer splits that result in large numbers of partitions, each being small but pure.

Gain Ratio



- Gain Ratio:

$$GainRATIO = \frac{GAIN}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

  - Parent Node's data is split into k partitions
  - n_i is the number of records in partition i; n = total number of records
  - Gain = reduction in entropy due to the test.
- Adjust Information Gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5
- Designed to overcome the disadvantage of Information Gain

Classification Error

- Given a set S:

$$Error(S) = 1 - \max_i P_i$$

- Measures misclassification error made by a node.
  - Maximum $(1 - 1/n_c)$
  - Minimum (0.0)

Example (Classification Error)

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

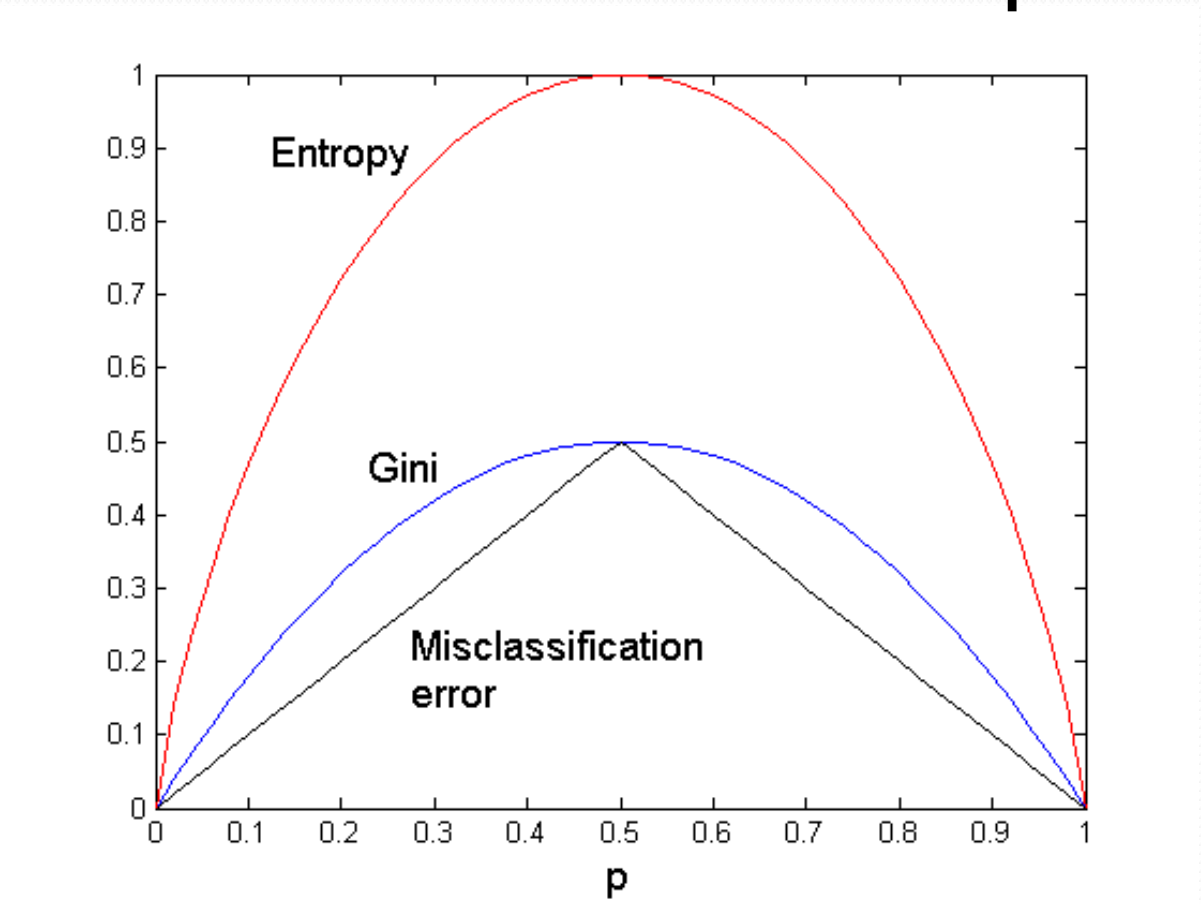
C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison

For a 2-class problem:



fraction of records in one of the two classes

Evaluating Classifiers

- Accuracy metrics
- How to divide labeled data into a training set and a test set?

Accuracy metrics

- Confusion Matrix:

2 classes:
"Yes": positive (or target) class
"No": negative class

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	*a*	*b*
	Class=No	*c*	*d*

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

correct

incorrect

Accuracy

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Error rate (e) = 1 - accuracy

Classification

- The simple “accuracy” measure may be misleading especially when the classes are “imbalanced”.
- Example:
  - two class labels: cancer (+ve) / no cancer (-ve)
  - only 1% of test records are with cancer
  - a simple classifier: just say “no cancer”
  - classification accuracy 99%

Other metrics

- Assume two class labels (positive, negative)
- precision = TP/(TP + FP)
  - The fraction of "positive class predictions" that are truly positives.
- recall = TP / (TP + FN)
  - The fraction of "positives" that are predicted positives.
- F measure = (2 * precision * recall) / (precision + recall)
- sensitivity (true positive rate) = recall
- specificity (true negative rate) = TN / (TN + FP)
- ROC curves

Getting a (training set, test set) pair

- Holdout
- Random subsampling
- Cross validation
- Bootstrap
- *Re-substitution (training set = test set)* ✘

Holdout Method

- Given a set of *N* labeled records (examples)
- Reserve a fraction (e.g., 80%) for training and use the rest for testing
- Limitations
  - Fewer labeled examples available for training
  - Not all data is involved in model training; not all data is involved in model testing.
  - Larger training set ⇒ more accurate model built, but fewer test examples to give a good accuracy evaluation … (and vise versa)

Random subsampling

- Repeated holdout
- Repeat experiment *k* times, each with a different sample of the data as holdout
- Classifier's accuracy = average accuracy of classifier on samples
- Limitations
  - some records may be used more often than others in training/testing

Cross-validation

- *k*-fold cross-validation:
  - Divide the examples into *k* partitions
  - Run classifier by using *k*-1 partitions as training data and one partition as test data
  - Repeat for all combinations of *k*-1 partitions and measure average accuracy
- leave-one-out: special case for *k*=*N*
  - Can be more accurate but computationally expensive

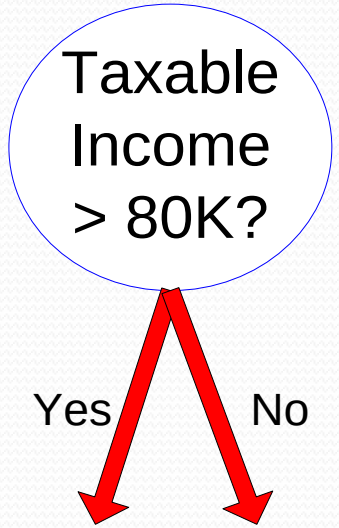
Bootstrap approach

- Given *N* examples, draw *N* times with replacement
- An *N*-sized sample contains (on average) 63.2% of the data
  - Prob. that an example is selected (for large *N*):
  - $1 - \left(1 - \frac{1}{N}\right)^N \approx 1 - \frac{1}{e} = 0.632$
- Records not included in the bootstrap become the test set
- Repeat *b* times and compute average accuracy

Numerical attribute

- Binary test on a value ($A > v$?)
- Number of possible splitting values ~= Number of distinct values
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally expensive

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



Numerical attribute

- More efficient method:
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index

label	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →		65		72		80		87		92		97		110		122		172		
		<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	
Yes		0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	
No		1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	
Gini		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		

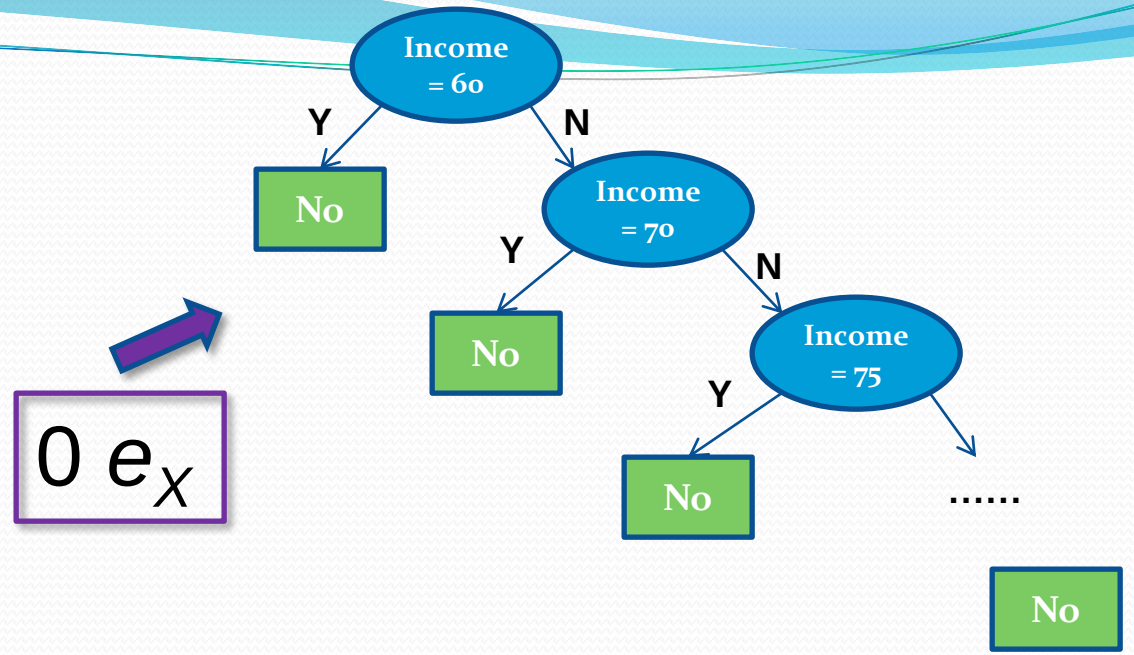
Training, test, and generalization errors

- Given a training set X, we derive a model M
- We can apply the model M on X and measure the error of the model in classifying X's records.
  - The error is called *training error* e_X
- We can apply the model M on a test set Y
  - The error is called *test error* e_Y
- *Generalization error* (e_G) = expected error when M is applied to a future unseen record.
- Since M is derived from X, it naturally fits X.
- In general, $e_G > e_X$
- We hope that $e_G \approx e_Y$

Overfitting and Underfitting

- When we build M, we aim at reducing the impurity of a tree node. Essentially, we are building a model M that reduces e_X.
- Issue: M may be fitting the training set too well that it becomes *too specific* to the training set. This issue is called *model overfitting*.
- Model overfitting $\Rightarrow$ M is not generally applicable to records other than the training set $\Rightarrow$ large e_G.
- The more complex the model is, the more likely that the model overfits the training data.

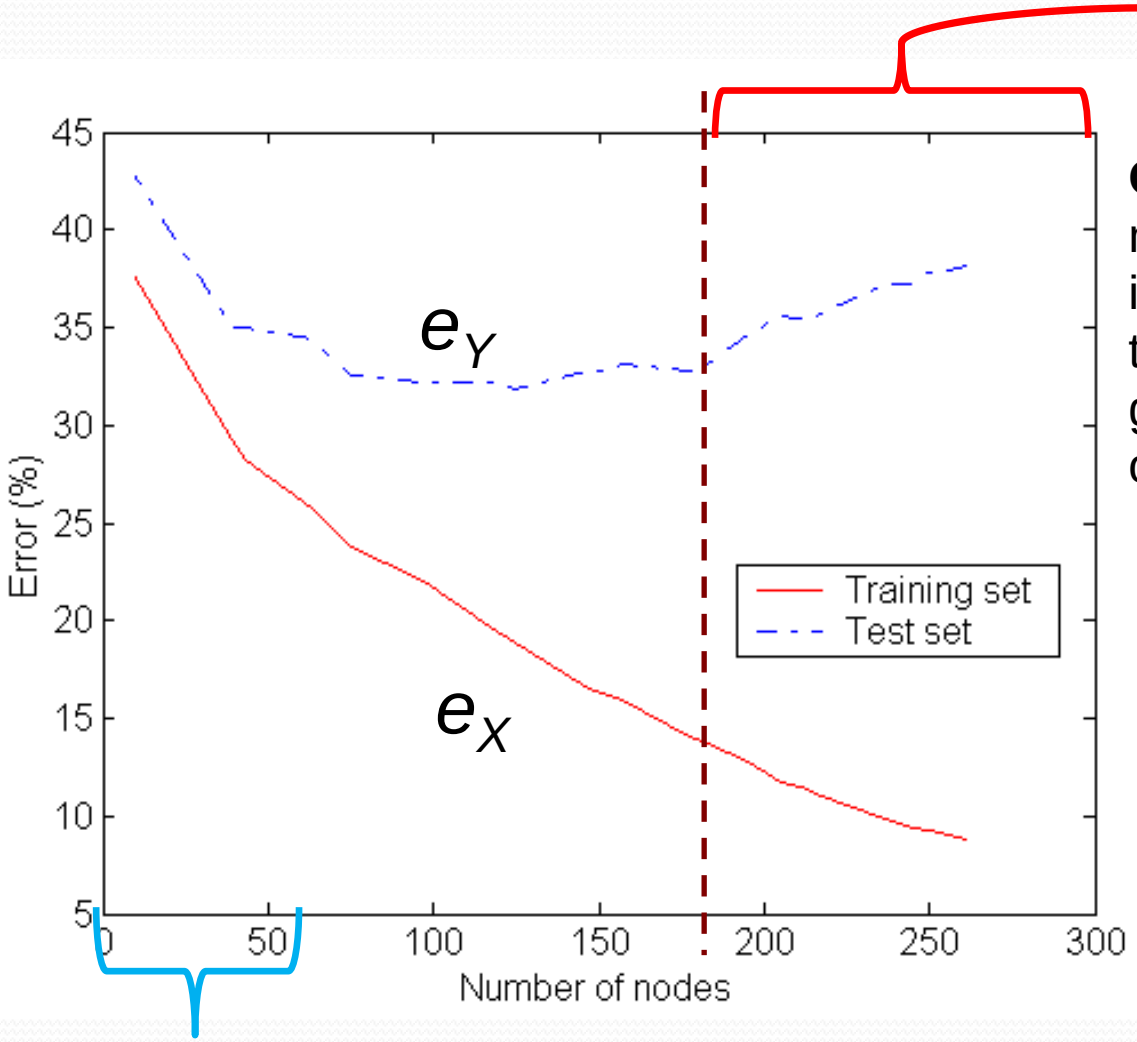
Overfitting



Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	***0.300***	0.343	0.375	0.400	0.420

Underfitting

- If the model *M* is too simple, then it probably won't fit even the training set. This is called *model underfitting*.



Overfitting: when the model is too complex, it is too specific to the training set and is not generally applicable to other data.

Underfitting: when model is too simple, both training and test errors are large

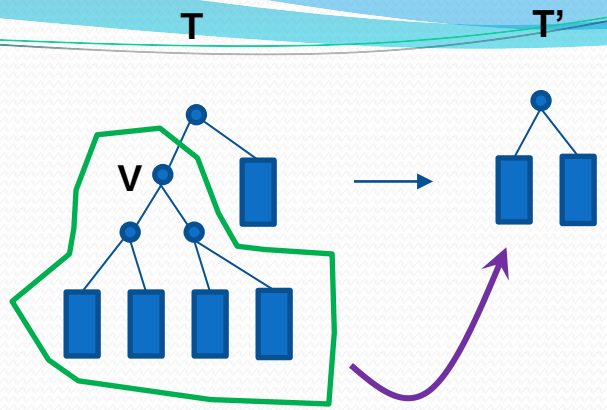
Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on future unseen records

Handling overfitting

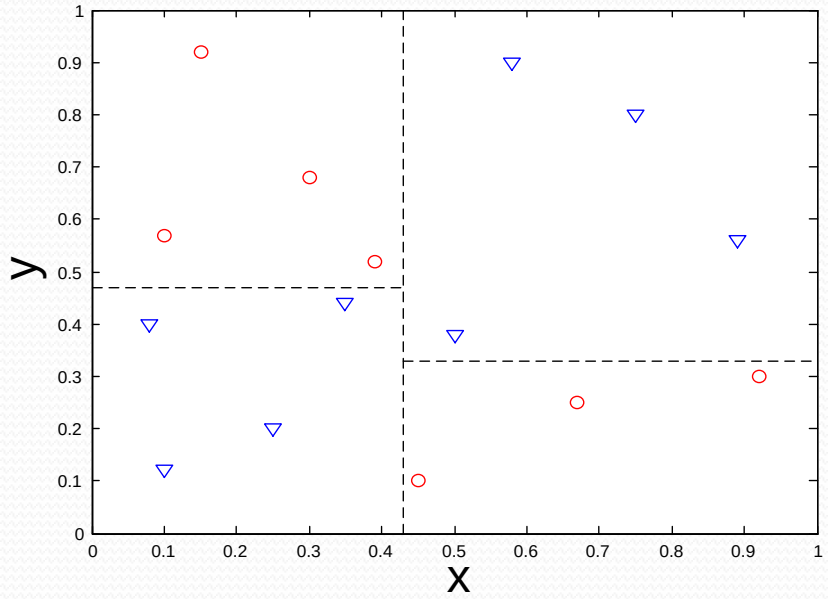
- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before the model becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More aggressive stopping conditions:
    - Stop if number of instances in a node is less than some user-specified threshold
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain) by much.

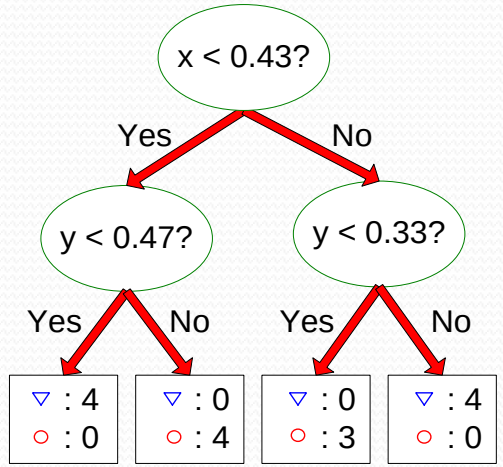
Handling overfitting



- Post-pruning
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - Class label of leaf node is determined from majority class of instances in the sub-tree
  - Basic idea: given a tree *T* and a node *v* to be pruned to obtain a pruned tree *T'*:
    - Estimate a cost complexity (cc): A function of tree size and error
      - More node: ↑ cc; Larger error: ↑ cc
    - If cc of *T'* < cc of *T*, prune *v*.

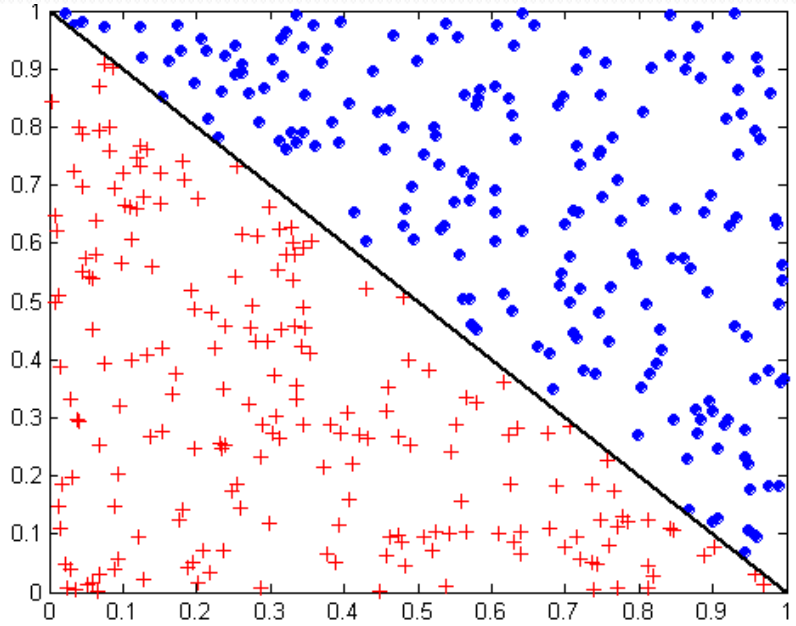
Decision Boundary

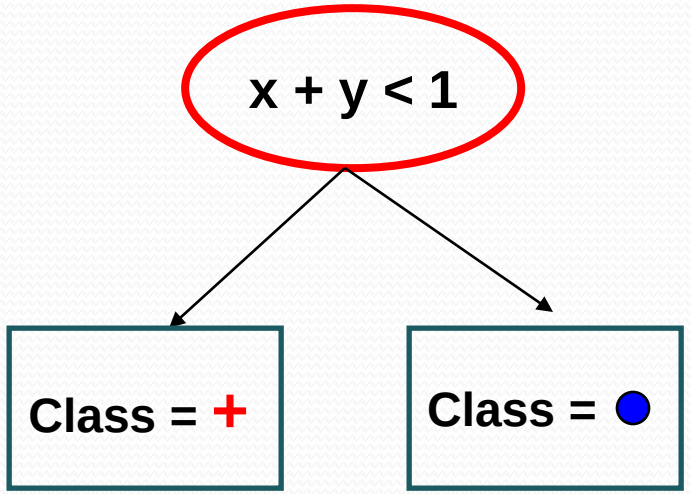




- Each test in a decision tree derives a *decision boundary*
- Decision boundary is parallel to axes because test condition involves a single attribute at-a-time

Oblique Decision Trees





- **Test condition may involve multiple attributes**

Decision Tree Based Classification

- Advantages:
  - Non-parametric
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Interpretable models
  - Accuracy is comparable to other classification techniques for many simple data sets
- Disadvantages
  - Subject to overfitting
  - Decision boundaries are only rectilinear