Chapter 8

Classification (Part 1)

Xike Xie

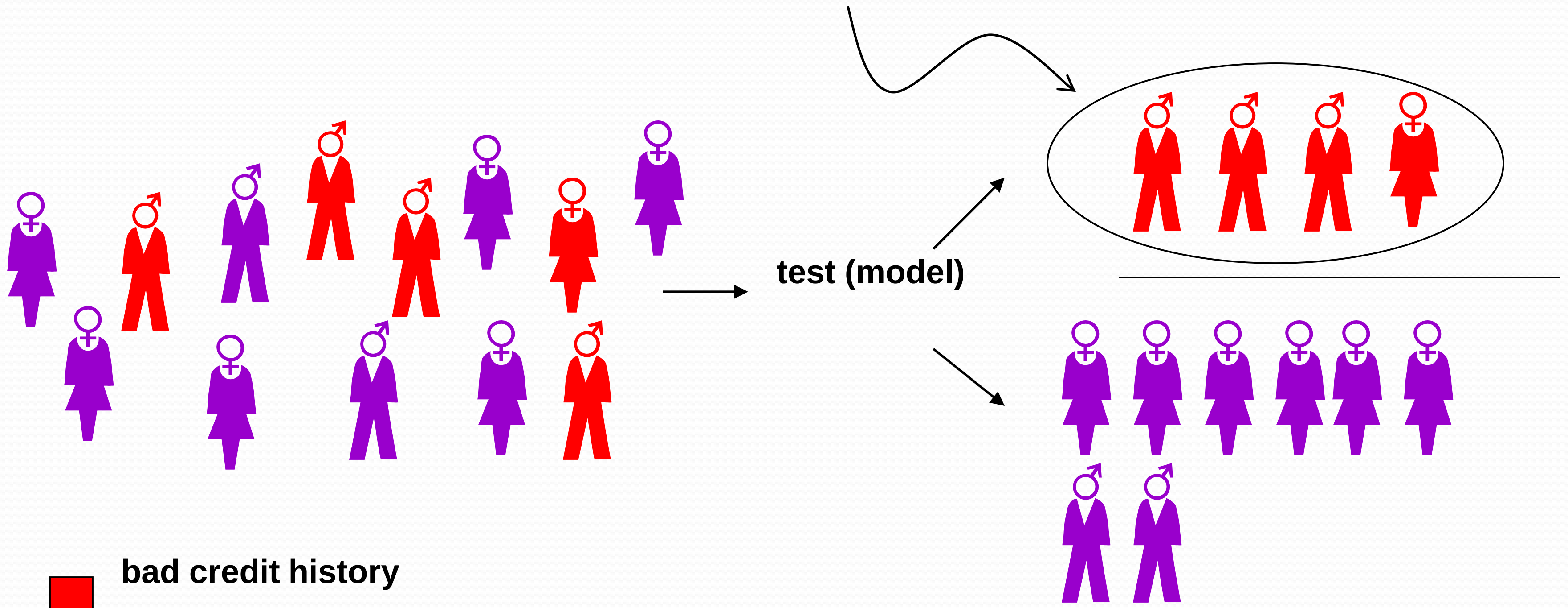
Slides are based on Prof. Ben Kao's work.

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?



bad credit history

good credit history

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

Data Model

- a dataset consists of a number of records
- each record consists of a number of attribute values
- one particular attribute is called the *label* (or *class*)
- 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 should belong

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*.

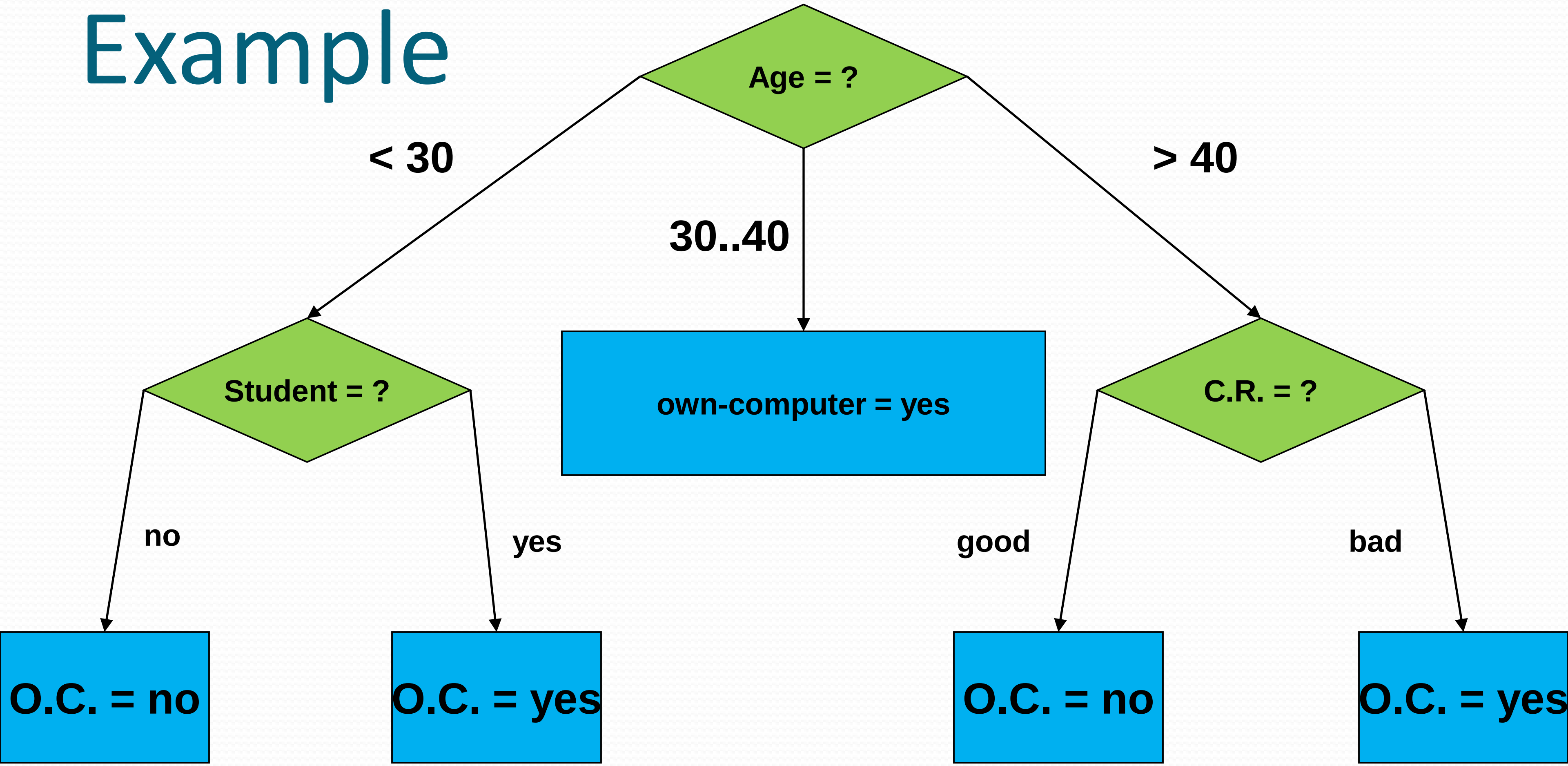
Decision-Tree Classifier (DT)

- 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

- entropy is a measure of *uncertainty* or *randomness*
- consider an information source which sends a stream of symbols (‘*a*’ or ‘*b*’) to a recipient

Entropy

- Q: how certain is the recipient in *predicting* the next symbol the information source would send?
- A: depends on the *probabilities* with which the source sends 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 *numeric measure* to quantify the concept of *uncertainty*
- According to Shannon, the entropy of an information source, S, is defined as:

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

  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

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

Very uncertain

  - ‘a’ (1.0), ‘b’ (0.0)
    - entropy = 1.0 * $\log_2$ (1/1.0) = 0

Very certain

A bit uncertain

  - ‘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 very uncertain

Entropy

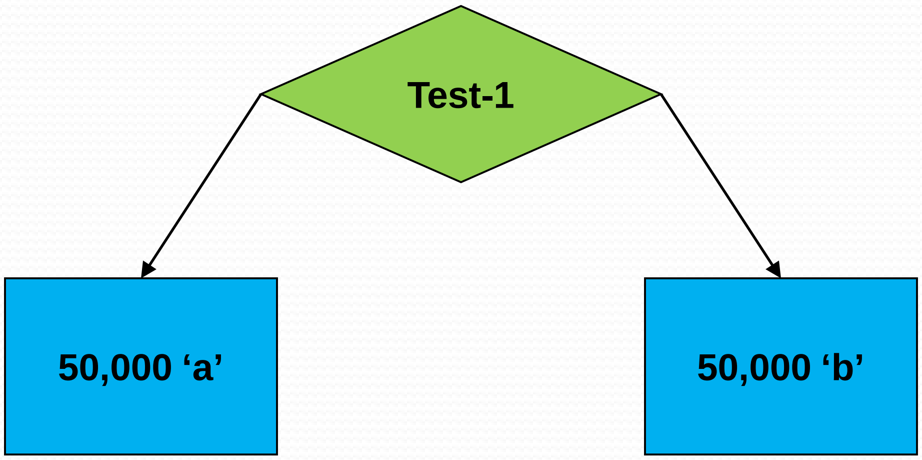
- the concept of entropy is used to select the tests used in a decision tree
- consider a dataset of 100,000 records with
  - 50,000 labeled ‘a’
  - 50,000 labeled ‘b’
  - if you are given a new record (the 100,001st) and you are asked to give it a label, how uncertain are you?
  - using entropy, your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

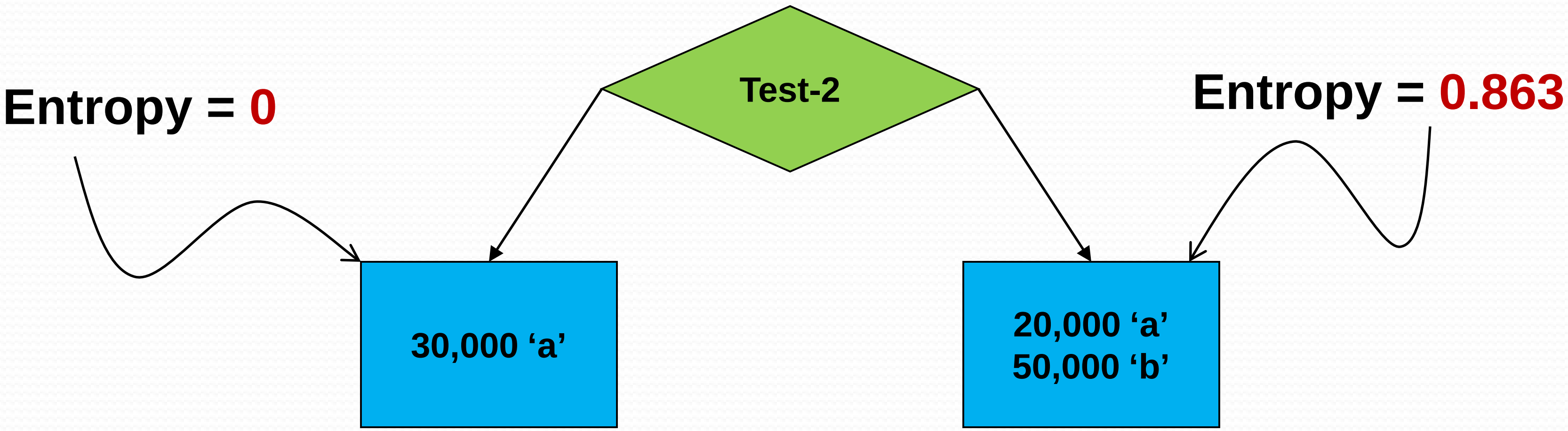
- How good is Test-1?

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

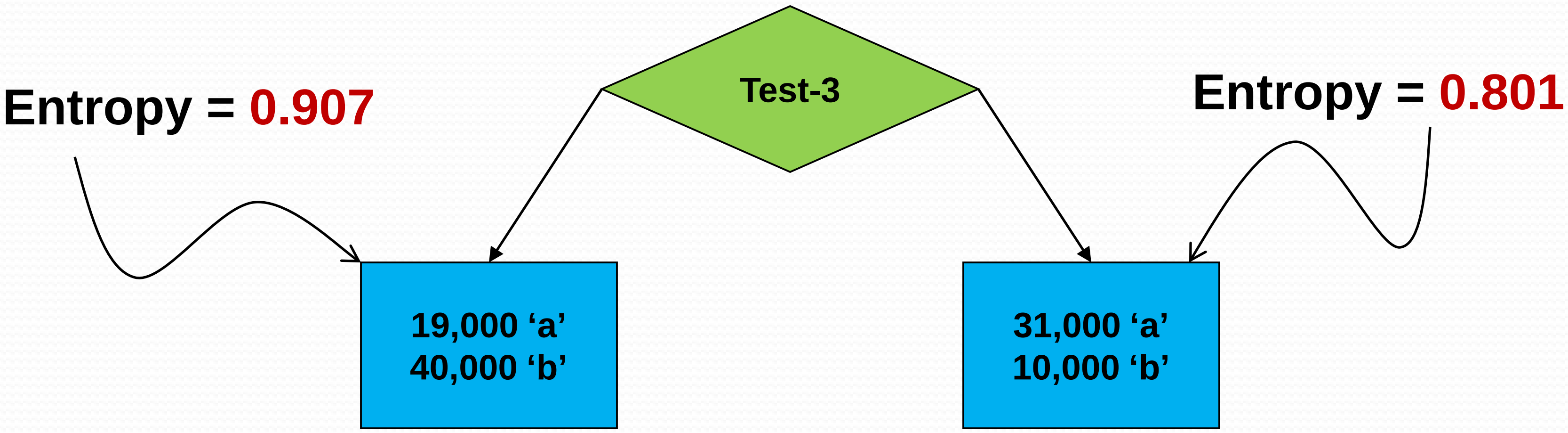


Finding a Good Test

- since 30% of the records are put to the left group by Test2 (and 70% to the right), given a new record, *x*, we would expect that *x* will go to the left group (after Test2) with a 30% probability
- the expected entropy (after Test2) is thus:
  - 0.3 * 0 + 0.7 * 0.863 = 0.604
- Test2 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



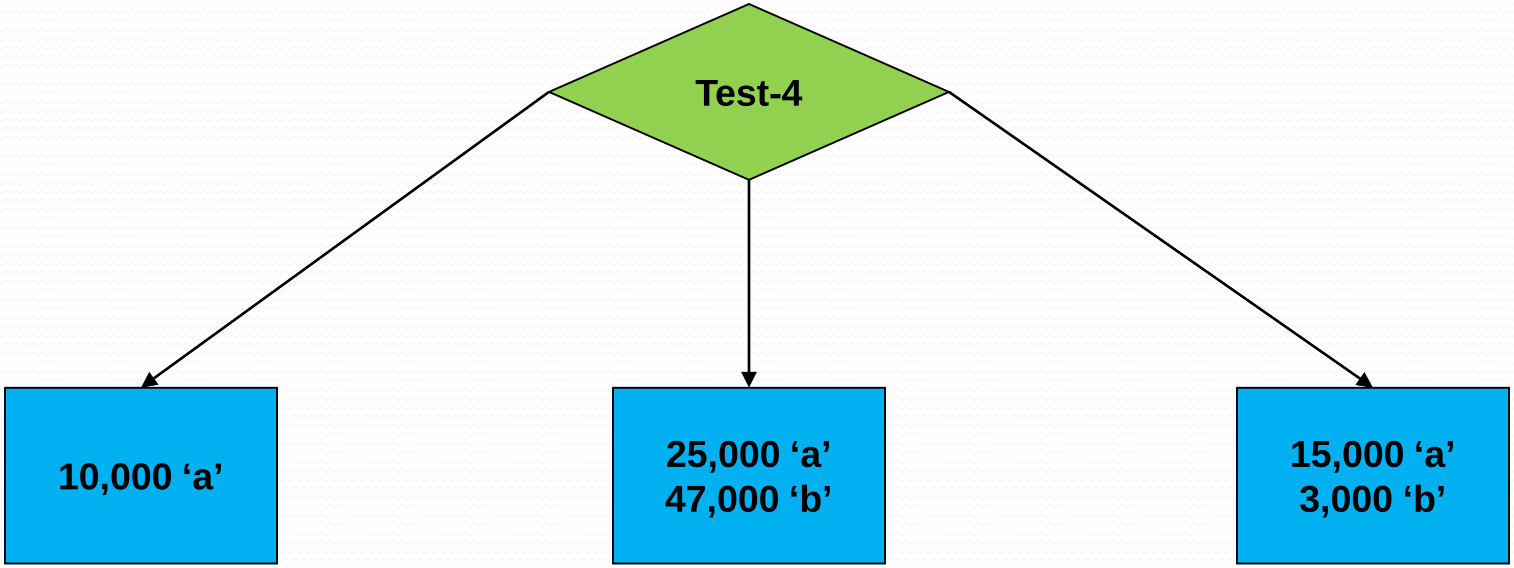
- expected entropy after Test-3:
  - (59/100) * 0.907 + (41/100) * 0.801 = 0.864

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

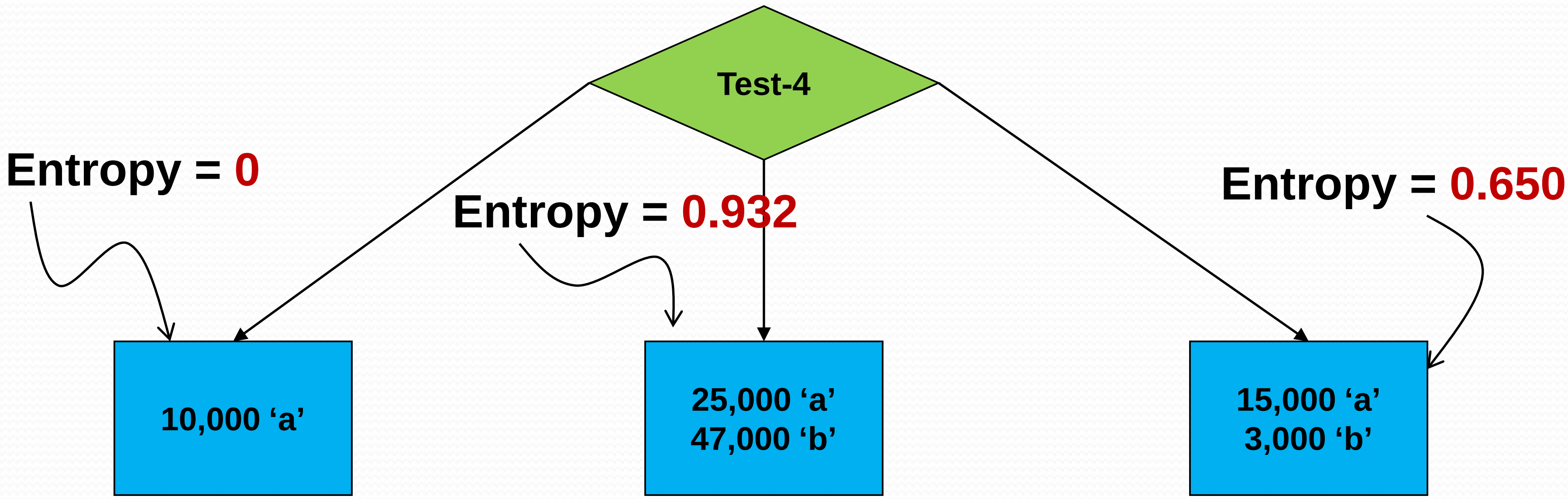
Finding a Good Test

- 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

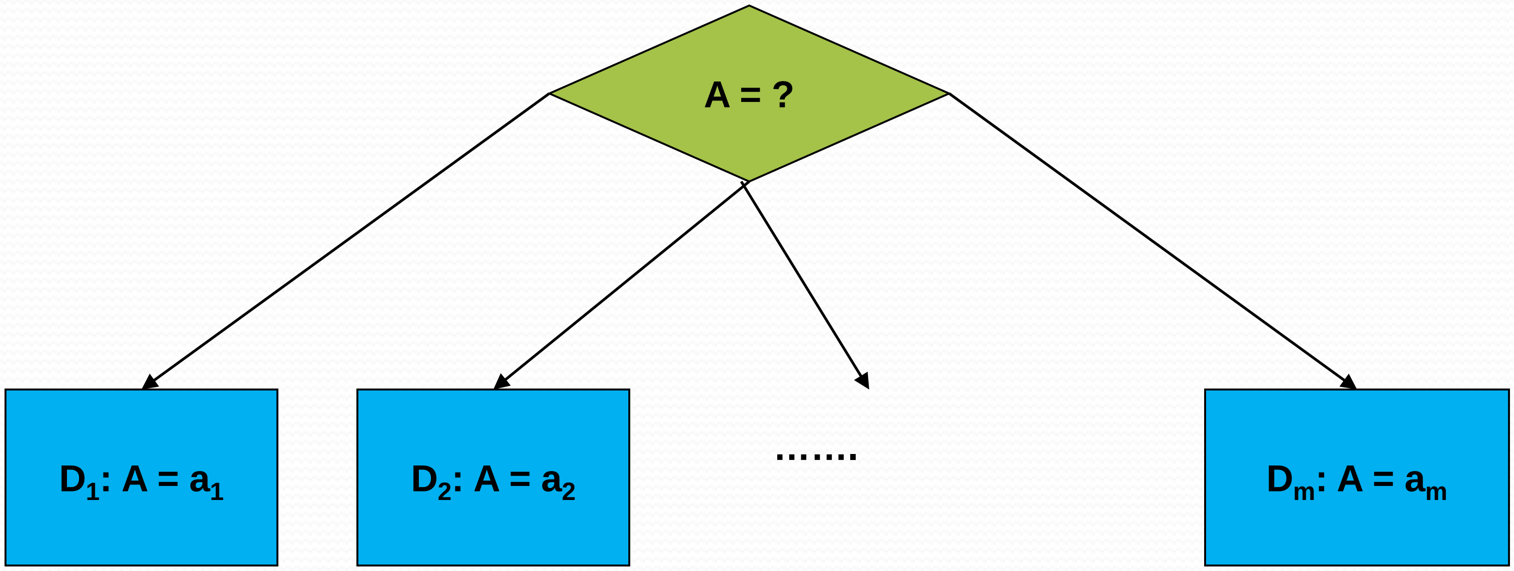
- base on the concept of test and entropy
- call our algorithm build-tree(D,L) where
  - D is a training set of data records
  - L is a list of nominal attributes
- want to build a decision tree using tests on attributes' values
- A recursive construction algorithm

Procedure (base cases)

- if …
  - all records in D are of the same label K
    - build-tree(D,L) returns a tree with only a leaf node labeled K. No test is needed
  - L is empty $\Rightarrow$ 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 $L \Rightarrow$ no meaningful tests can be derived
    - repeat the action of the 2nd case.

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: *D1* … *Dm*
- create a node labeled "*A*=?"
- for each group, *Di*,
  - if *Di* is empty, create a subtree *Ti* with a lone leaf node labeled *M*, where *M* is the most common label among the records in *D*
  - else
    - recursively call build-tree(*Di*, *L*-{*A*}) to build a subtree *Ti*,
    - connect the test node "*A*=?" to *Ti* by a branch labeled "a_i"

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"
  - 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	Stude nt	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) =
  $(5/14 * 0.971) + (4/14 * 0) + (5/14 * 0.971) = \boxed{0.694}$

Example

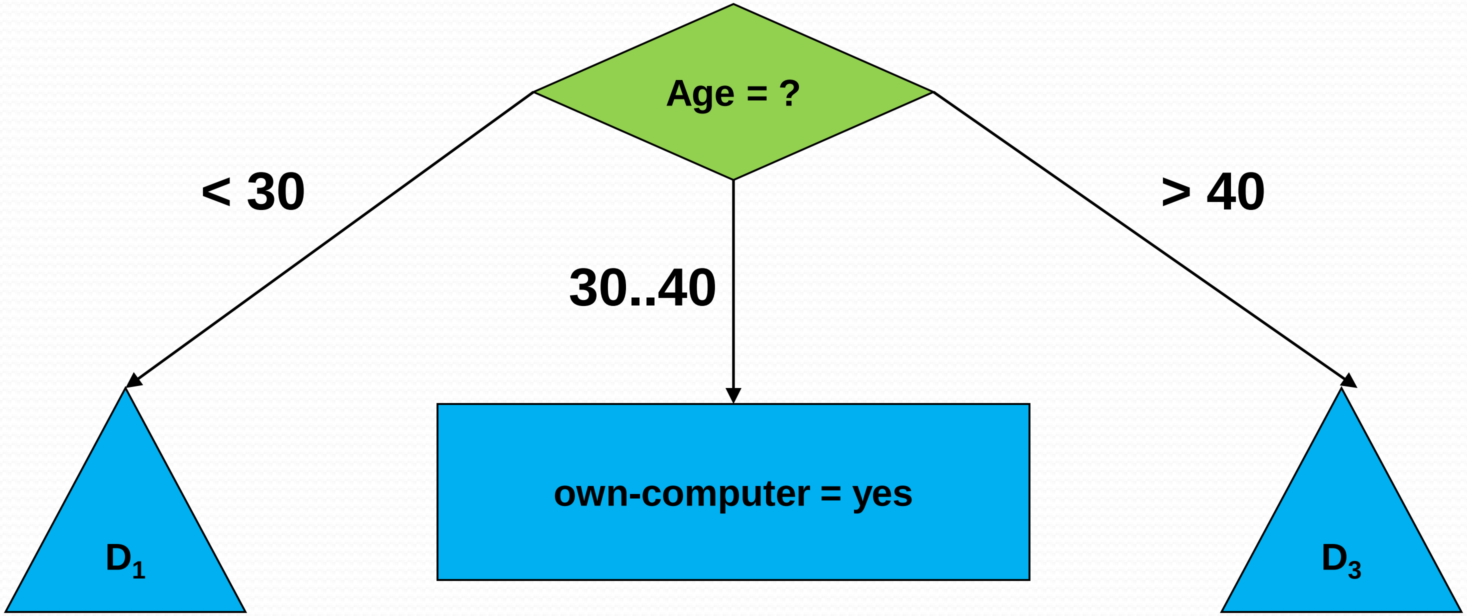
Best

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}
- we 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

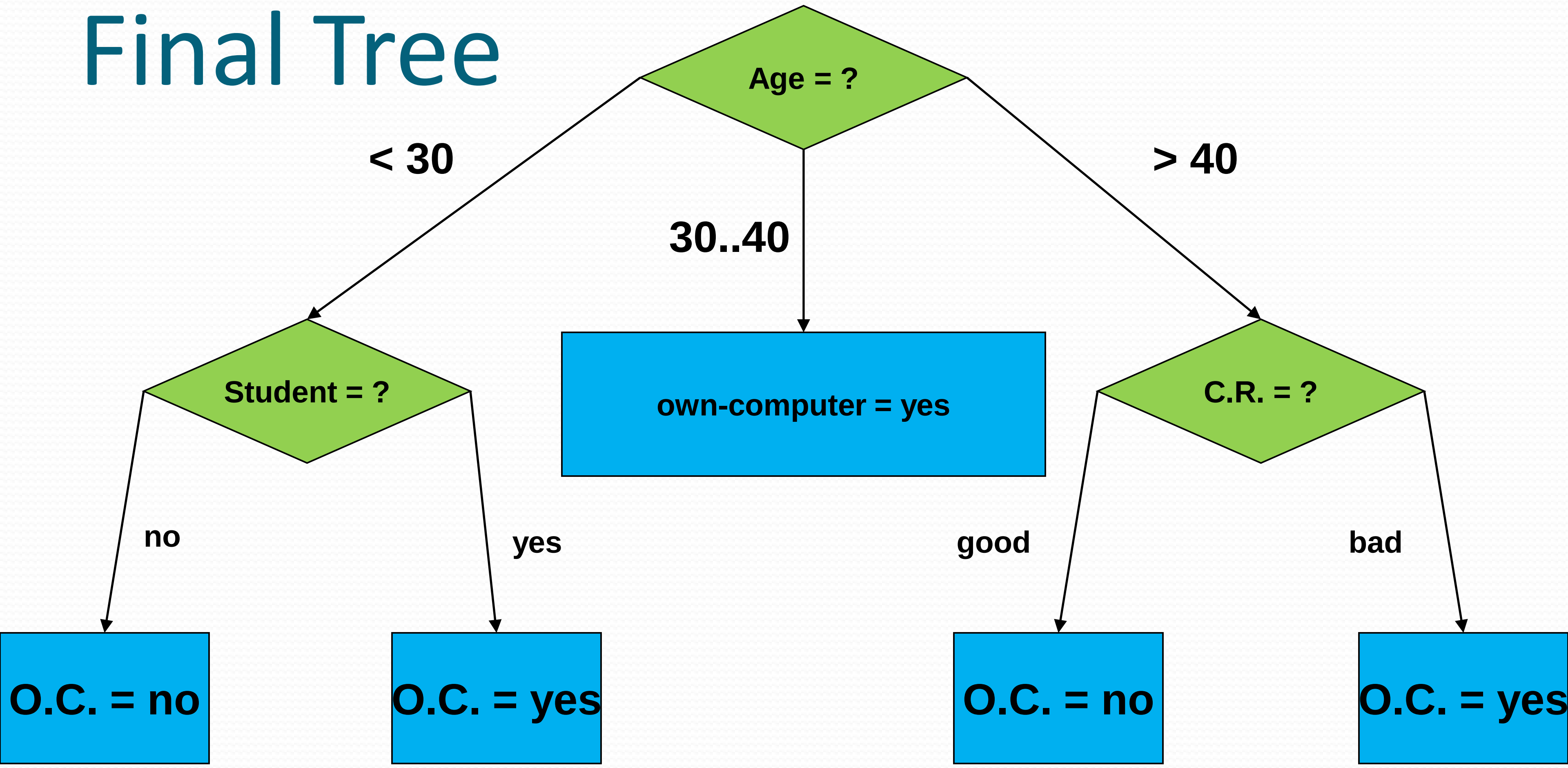
attribute test	expected entropy
Income	0.4
Student	0
Credit-rating	0.951

Example

- For $D3$, 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

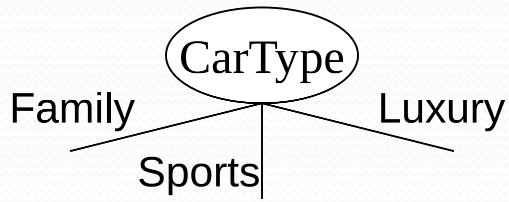
- How to derive a test based on an attribute?
- How to measure the purity (certainty, uncertainty) of a dataset?
- How to measure the performance of a classifier?
- Numerical attribute
- Overfitting

How to derive a test?

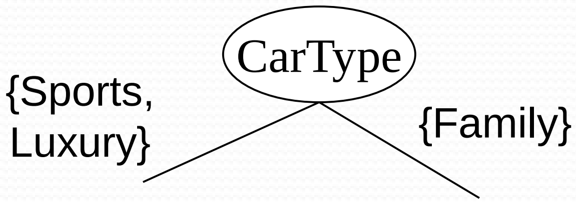
- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous (numerical)
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting (Nominal Attributes)

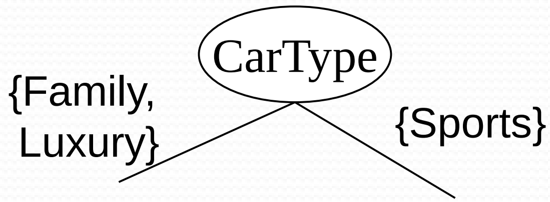
- *Multi-way split*: Use as many partitions as distinct values.



- *Binary split*: Divides values into two subsets. Need to find optimal partitioning.

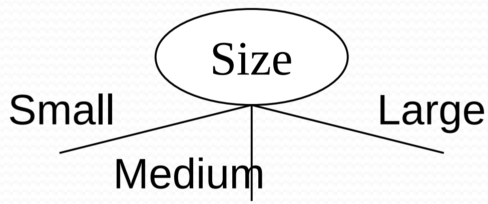


OR



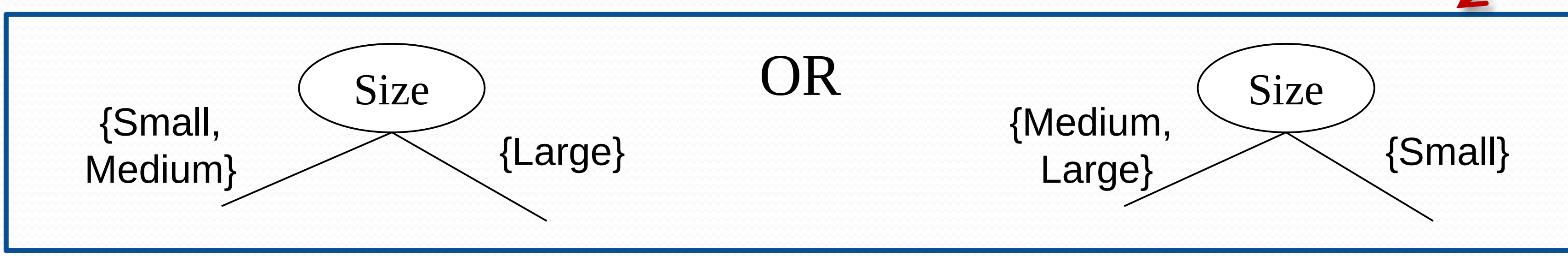
Splitting (Ordinal Attributes)

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

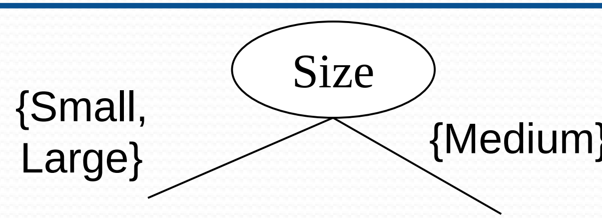


Order preserving

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



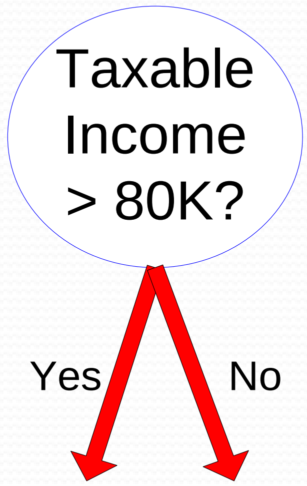
- What about this split?



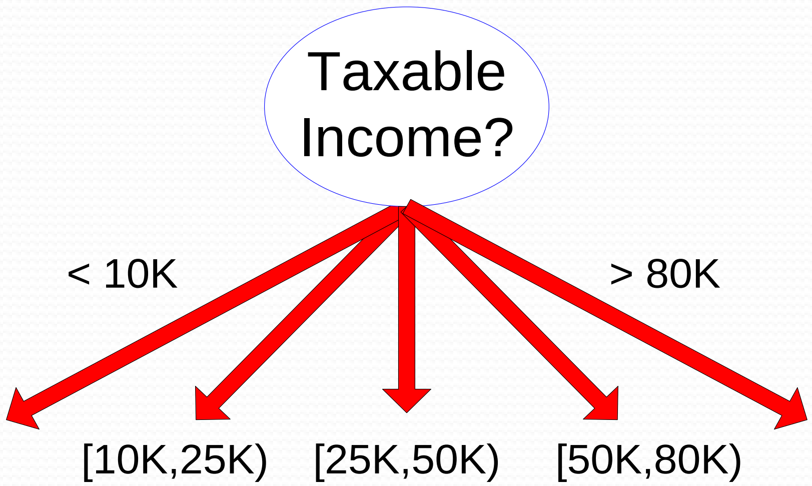
Splitting (Continuous Attributes)

- *Discretization*: transform it to an ordinal attribute
- *Binary Decision*: $(A < v)$ or $(A \geq v)$
  - consider all possible splits and finds the best cut
  - can be more computationally expensive

Splitting (Continuous Attributes)



(i) Binary split



(ii) Multi-way split

Impurity measures

- Entropy (Information Gain) – ID3
- Gain Ratio – C4.5
- Gini Index - CART
- Classification error

Entropy/Information gain

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

Gain Ratio

- Gain Ratio:

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

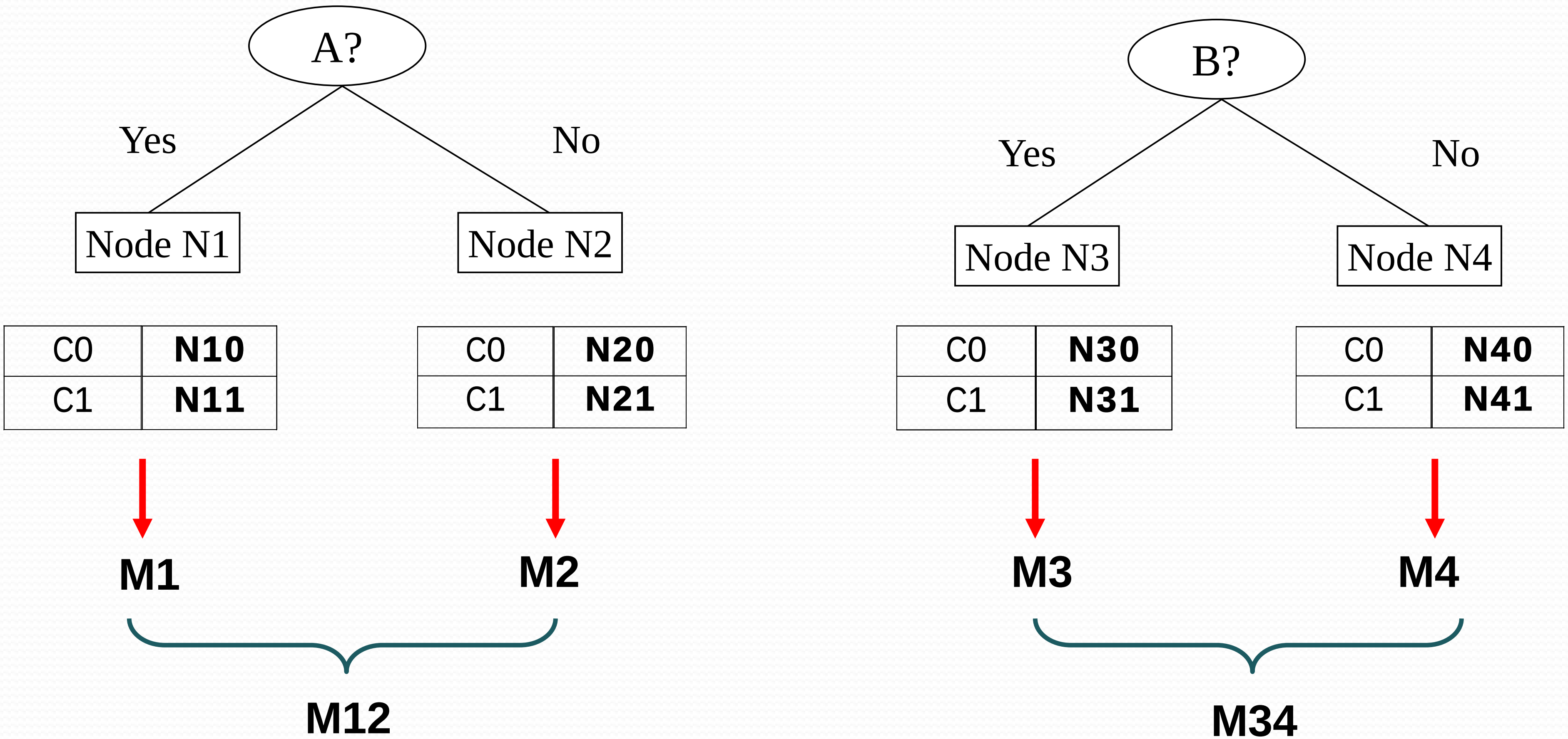
$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

  - Parent Node, p is split into k partitions
  - n_i is the number of records in partition i
  - Gain = reduction in entropy due to the test.
- Adjusts 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

Before Splitting:

C0	**N00**
C1	**N01**

→ **M0**



Gain = M0 – M12 vs M0 – M34

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

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

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

Example (GINI)

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

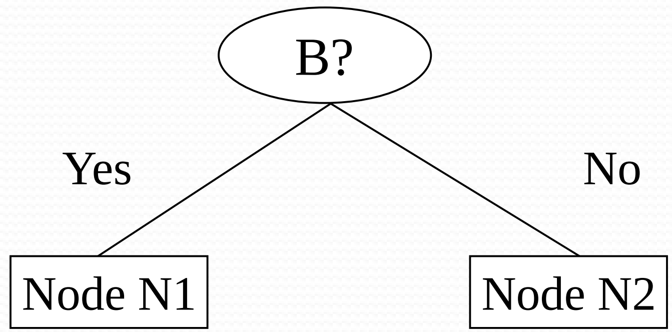
Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

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

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

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444

Example (GINI)



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

Gini(N1)
$= 1 - (5/7)^2 - (2/7)^2$
= 0.194

Gini(N2)
$= 1 - (1/5)^2 - (4/5)^2$
= 0.528

	N1	N2
C1	**5**	**1**
C2	**2**	**4**
Gini=0.333		

Gini(Children)
= 7/12 * 0.194 +
5/12 * 0.528
= 0.333

Classification Error

- Given a set S:

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

- Measures misclassification error made by a node, where n_c is the number of classes
  - 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

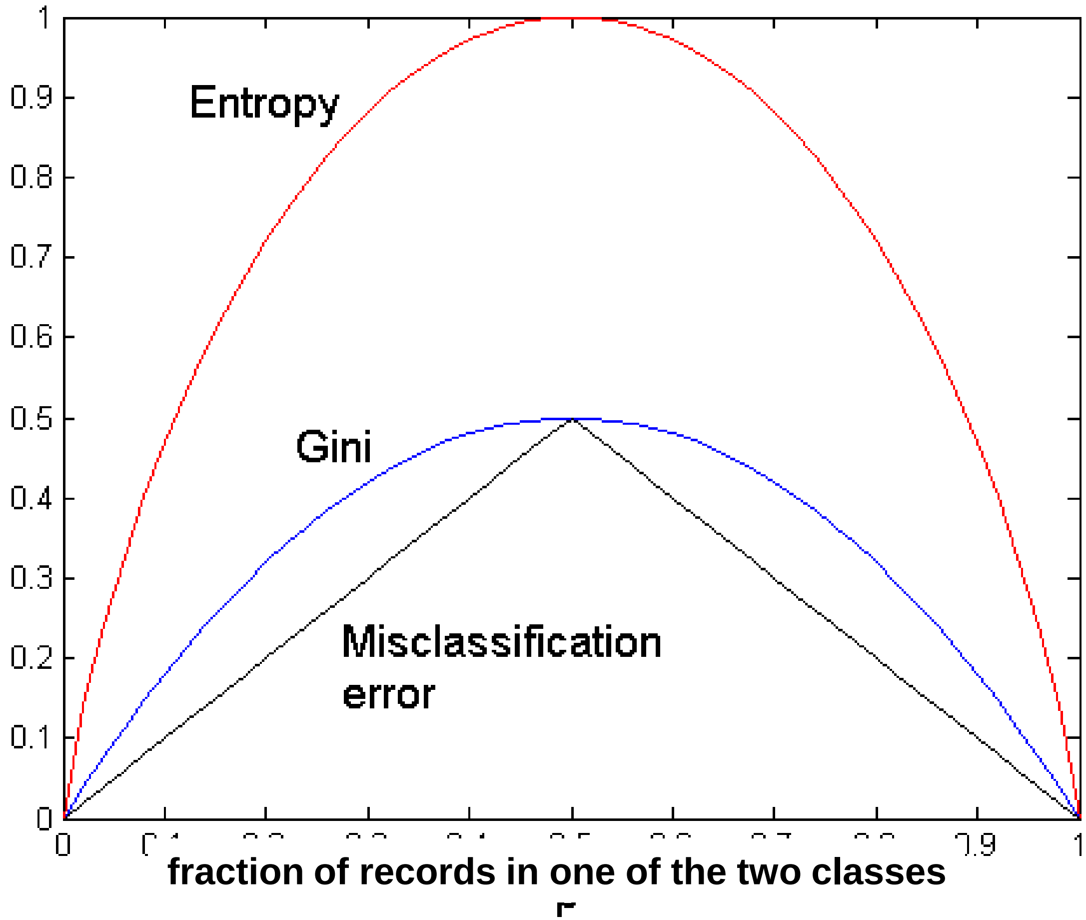
C1	**2**
C2	**4**

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:



Exercise

Q1. It is important to calculate the worst-case computational complexity of the decision tree algorithm. Given data set D, the number of attributes n, and the number of training tuples $|D|$, show that the computational cost of growing a tree is at most $n \times |D| \times log(|D|)$.

Evaluating Classifiers

- Accuracy metrics
- Training set vs. test set

Accuracy metrics

- Use test data of known class labels
- Confusion Matrix:

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

correct   *incorrect*

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

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

The simple "accuracy" measure may be misleading especially when the classes are "imbalanced".

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)

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

Getting a (training set, test set) pair

- Holdout
- Random subsampling
- Cross validation
- Bootstrap

Holdout Method

- Given a set of N labeled records (examples)
- Reserve 2/3 for training and 1/3 for testing
- Limitations
  - fewer labeled examples available for training
  - Larger training set $\Rightarrow$ 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

Bootstrap approach

- Sample with replacement
- An N-sized sample contains 63.2% of the data

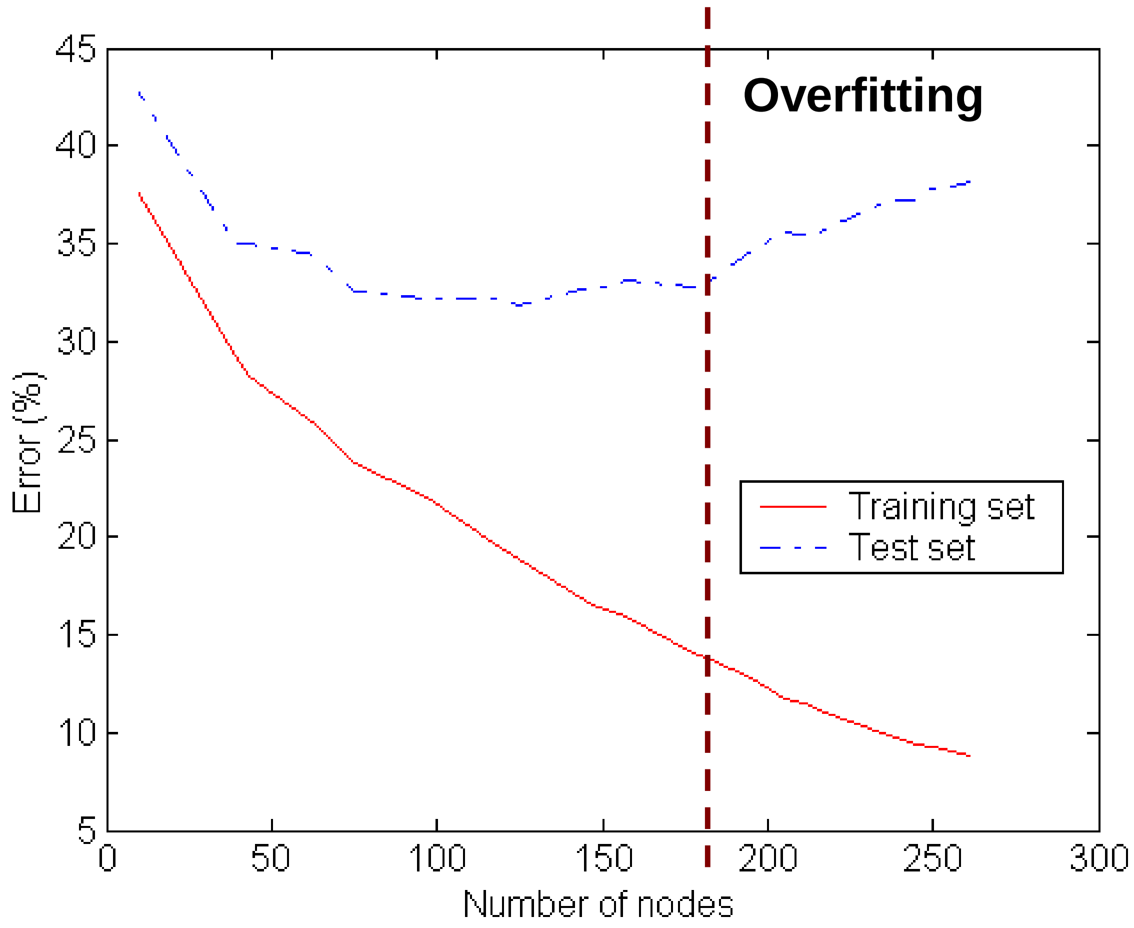
- Repeat b times and compute average accuracy

Training error vs. Generalization error

- 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 $e_G \approx e_Y$

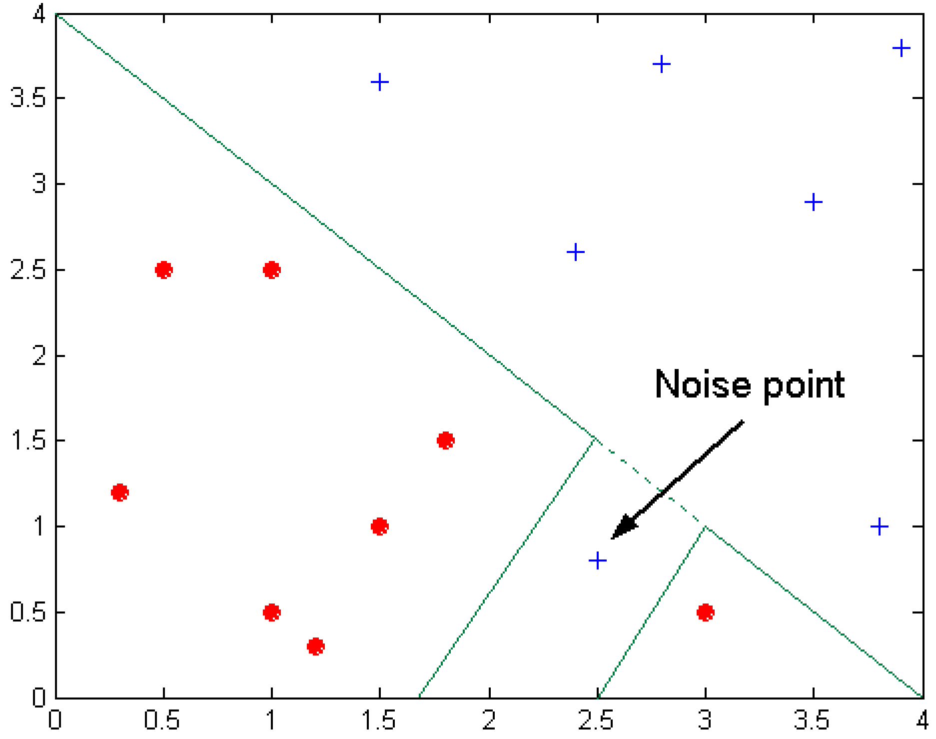
Overfitting and Underfitting

- When we build M, we aim at reducing the impurity of a tree node. Essentially, we are building an 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.



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

Overfitting due to noise



- **Overfitting results in decision trees that are more complex than necessary**

Decision boundary is distorted by noise point

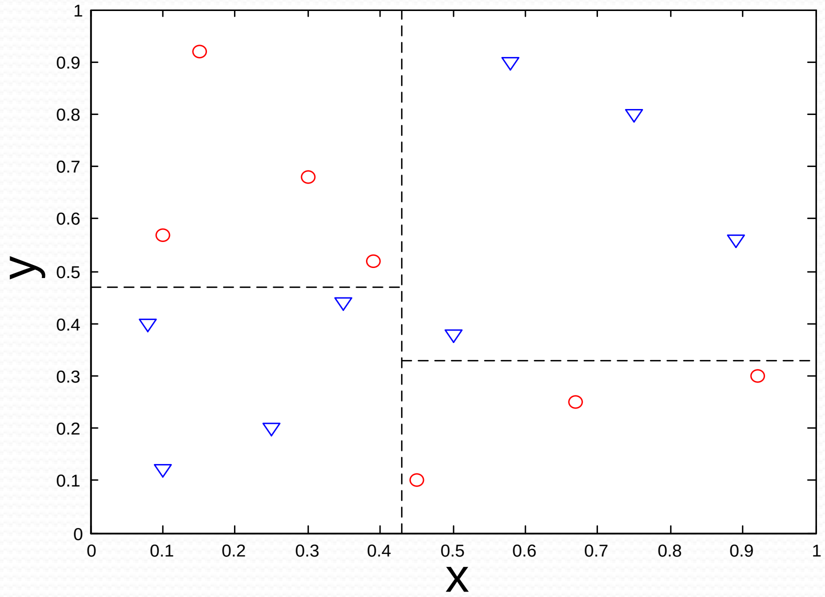
Handling overfitting

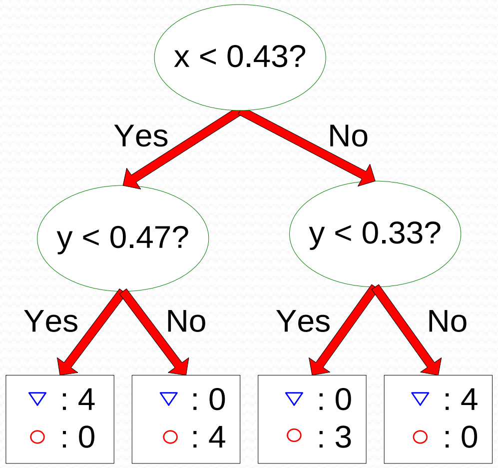
- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if **number of instances 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):
      - More node: ↑ cc; Larger error: ↑ cc
    - If cc of *T'* < cc of *T*, prune *v*.

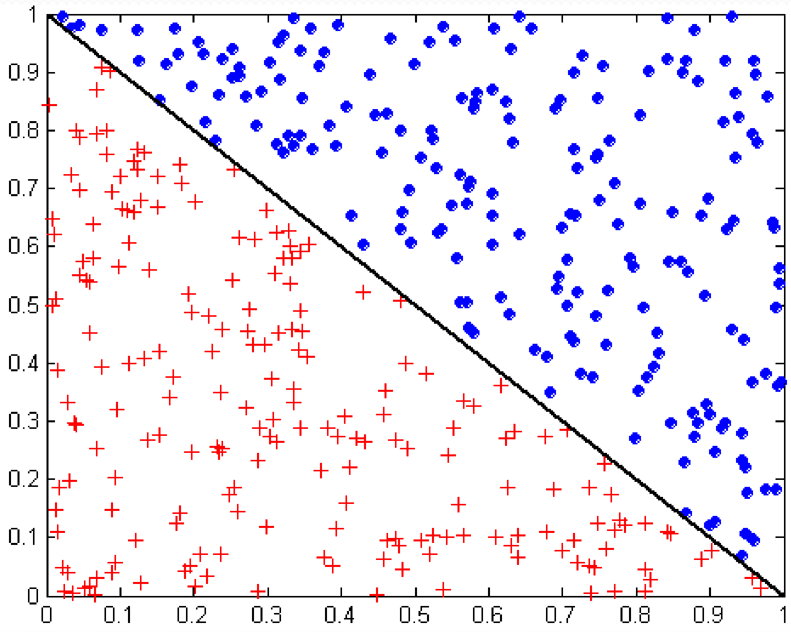
Decision Boundary

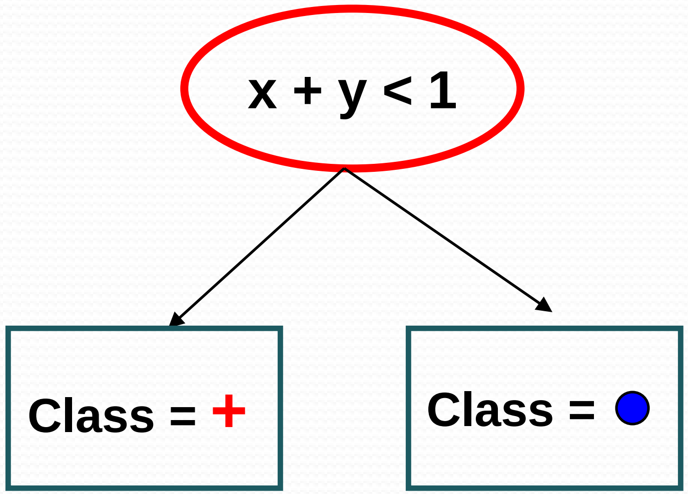




- **Border line between two neighboring regions of different classes is known as decision boundary**
- **Decision boundary is parallel to axes because test condition involves a single attribute at-a-time**

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
  - Easy to interpret small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets
- Disadvantages
  - Subject to overfitting
  - Do not generalize well for some discrete-valued functions (e.g. parity functions)
  - Decision boundaries are only rectilinear