OLAP and Data Mining

CSE 532, Theory of Database Systems

Stony Brook University

http://www.cs.stonybrook.edu/~cse532

OLTP Compared With OLAP

- On Line Transaction Processing – *OLTP*
  - Maintains a database that is an accurate model of some real-world enterprise. Supports day-to-day operations. Characteristics:
    - Short simple transactions
    - Relatively frequent updates
    - Transactions access only a small fraction of the database
- On Line Analytic Processing – *OLAP*
  - Uses information in database to guide strategic decisions. Characteristics:
    - Complex queries
    - Infrequent updates
    - Transactions access a large fraction of the database
    - Data need not be up-to-date

The Internet Grocer

- OLTP-style transaction:
  - John Smith, from Schenectady, N.Y., just bought a box of tomatoes; charge his account; deliver the tomatoes from our Schenectady warehouse; decrease our inventory of tomatoes from that warehouse
- OLAP-style transaction:
  - How many cases of tomatoes were sold in all northeast warehouses in the years 2000 and 2001?

OLAP: Traditional Compared with Newer Applications

- Traditional OLAP queries
  - Uses data the enterprise gathers in its usual activities, perhaps in its OLTP system
  - Queries are ad hoc, perhaps designed and carried out by non-professionals (managers)
- Newer Applications (e.g., Internet companies)
  - Enterprise actively gathers data it wants, perhaps purchasing it
  - Queries are sophisticated, designed by professionals, and used in more sophisticated ways

The Internet Grocer

- Traditional
  - How many cases of tomatoes were sold in all northeast warehouses in the years 2000 and 2001?
- Newer
  - Prepare a profile of the grocery purchases of John Smith for the years 2000 and 2001 (so that we can customize our marketing to him and get more of his business)

Data Mining

- *Data Mining* is an attempt at knowledge discovery – to extract knowledge from a database
- Comparison with OLAP
  - *OLAP*:
    - What percentage of people who make over $50,000 defaulted on their mortgage in the year 2000?
  - *Data Mining*:
    - How can information about salary, net worth, and other historical data be used to *predict* who will default on their mortgage?

Data Warehouses

- OLAP and data mining databases are frequently stored on special servers called ***data warehouses***:
  - Can accommodate the huge amount of data generated by OLTP systems
  - Allow OLAP queries and data mining to be run off-line so as not to impact the performance of OLTP

OLAP, Data Mining, and Analysis

- The “A” in OLAP stands for “Analytical”
- Many OLAP and Data Mining applications involve sophisticated analysis methods from the fields of mathematics, statistical analysis, and artificial intelligence
- Our main interest is in the database aspects of these fields, not the sophisticated analysis techniques

Fact Tables

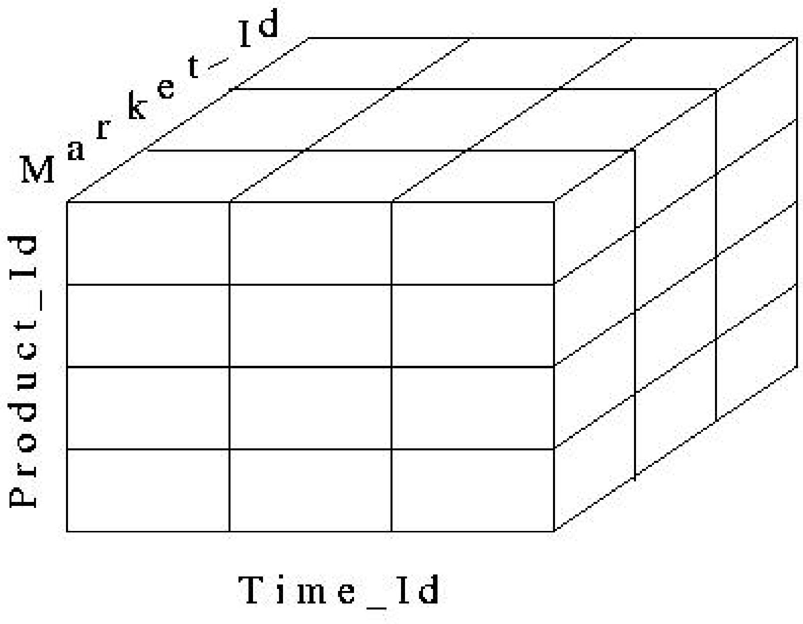
- Many OLAP applications are based on a ***fact table***
- For example, a supermarket application might be based on a table

  Sales (*Market_Id*, *Product_Id*, *Time_Id*, *Sales_Amt*)

- The table can be viewed as *multidimensional*
  - *Market_Id*, *Product_Id*, *Time_Id* are the dimensions that represent specific supermarkets, products, and time intervals
  - *Sales_Amt* is a function of the other three

A Data Cube

- Fact tables can be viewed as an N-dimensional *data cube* (3-dimensional in our example)
  - The entries in the cube are the values for *Sales_Amts*



Dimension Tables

- The dimensions of the fact table are further described with ***dimension tables***
- Fact table:

  Sales (*Market_id*, *Product_Id*, *Time_Id*, *Sales_Amt*)

- Dimension Tables:

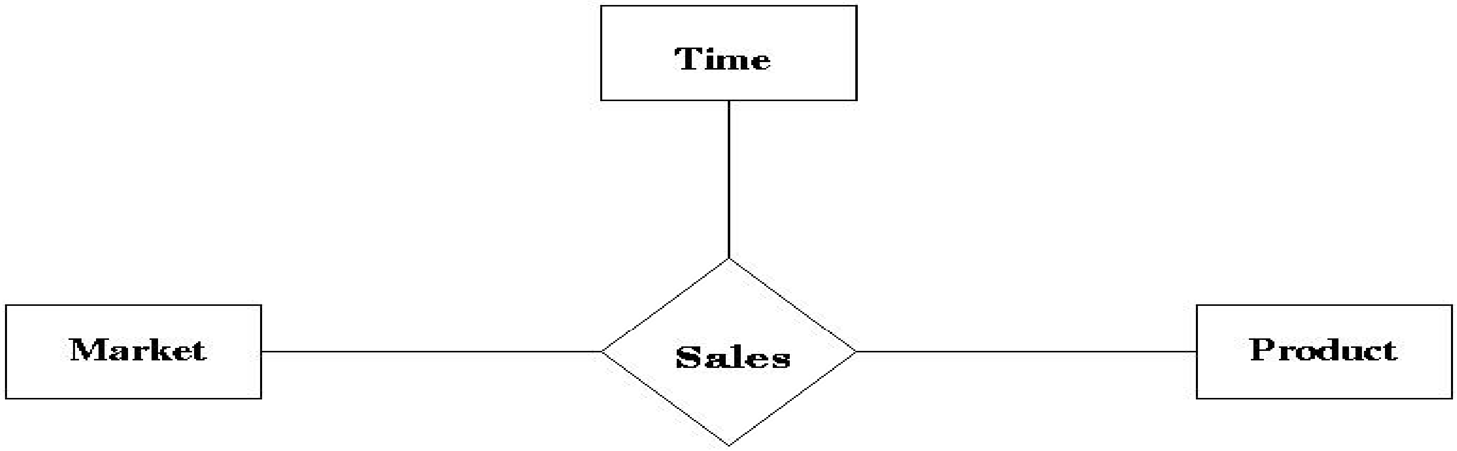
  Market (*Market_Id*, *City*, *State*, *Region*)

  Product (*Product_Id*, *Name*, *Category*, *Price*)

  Time (*Time_Id*,*Week*, *Month*, *Quarter*)

Star Schema

- The fact and dimension relations can be displayed in an E-R diagram, which looks like a star and is called a ***star schema***



Aggregation

- Many OLAP queries involve ***aggregation*** of the data in the fact table
- For example, to find the total sales (over time) of each product in each market, we might use

```
SELECT      S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM         Sales  S
GROUP BY  S.Market_Id, S.Product_Id
```

- The aggregation is over the entire time dimension and thus produces a two-dimensional view of the data. (Note: aggregation here is over time, not supermarkets or products.)

Aggregation over Time

- The output of the previous query

		Market_Id			
	SUM(*Sales_Amt*)	M1	M2	M3	M4
Product_Id	P1	3003	1503	…	
	P2	6003	2402	…	
	P3	4503	3	…	
	P4	7503	7000	…	
	P5	…	…	…	

Drilling Down and Rolling Up

- Some dimension tables form an ***aggregation hierarchy***

  Market_Id → *City* → *State* → *Region*

- Executing a series of queries that moves down a hierarchy (*e.g.*, from aggregation over regions to that over states) is called ***drilling down***
  - Requires the use of the fact table or information more specific than the requested aggregation (*e.g.*, cities)
- Executing a series of queries that moves up the hierarchy (e.g., from states to regions) is called ***rolling up***
  - Note: In a rollup, coarser aggregations can be computed using prior queries for finer aggregations

Drilling Down

- Drilling down on market: from *Region* to *State*

Sales (*Market_Id, Product_Id, Time_Id, Sales_Amt*)

Market (*Market_Id, City, State, Region*)

1.
```
SELECT      S.Product_Id, M.Region, SUM (S.Sales_Amt)
FROM        Sales  S,  Market  M
WHERE       M.Market_Id = S.Market_Id
GROUP BY  S.Product_Id,  M.Region
```

2.
```
SELECT      S.Product_Id, M.State, SUM (S.Sales_Amt)
FROM        Sales  S,  Market  M
WHERE       M.Market_Id = S.Market_Id
GROUP BY  S.Product_Id,  M.State,
```

Rolling Up

- Rolling up on market, from *State* to *Region*
  - If we have already created a table, State_Sales, using

```
1.    SELECT      S.Product_Id, M.State, SUM (S.Sales_Amt)
      FROM        Sales S, Market M
      WHERE       M.Market_Id = S.Market_Id
      GROUP BY    S.Product_Id, M.State
```

then we can roll up from there to:

```
2.    SELECT      T.Product_Id, M.Region, SUM (T.Sales_Amt)
      FROM        State_Sales T, Market M
      WHERE       M.State = T.State
      GROUP BY    T.Product_Id, M.Region
```

Can reuse the results of query 1.

Pivoting

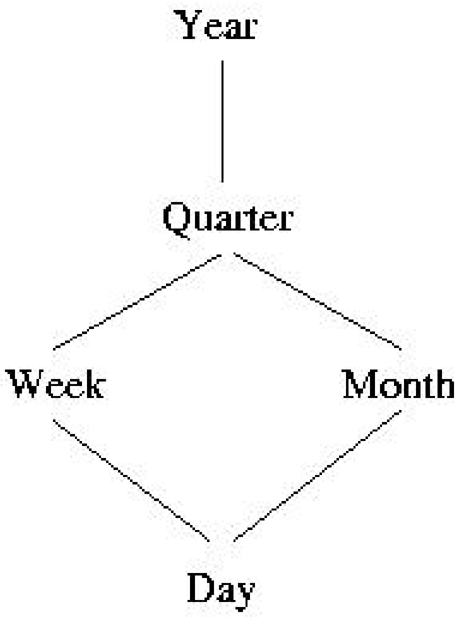
- When we view the data as a multi-dimensional cube and group on a subset of the axes, we are said to be performing a *pivot* on those axes
  - Pivoting on dimensions $D_1,\ldots,D_k$ in a data cube $D_1,\ldots,D_k,D_{k+1},\ldots,D_n$ means that we use GROUP BY $A_1,\ldots,A_k$ and aggregate over $A_{k+1},\ldots A_n$, where A_i is an attribute of the dimension D_i
  - *Example*: Pivoting on Product and Time corresponds to grouping on *Product_id* and *Quarter* and aggregating *Sales_Amt* over *Market_id:*

```
SELECT      S.Product_Id, T.Quarter,  SUM (S.Sales_Amt)
FROM         Sales S,  Time T
WHERE       T.Time_Id = S.Time_Id
GROUP BY  S.Product_Id,  T.Quarter
```

Pivot

Time Hierarchy as a Lattice

- Not all aggregation hierarchies are linear
  - The time hierarchy is a lattice
    - Weeks are not contained in months
    - We can roll up days into weeks or months, b ks into quarters



Slicing-and-Dicing

- When we use WHERE to specify a particular value for an axis (or several axes), we are performing a ***slice***
  - Slicing the data cube in the Time dimension (choosing sales only in week 12) then pivoting to *Product_id* (aggregating over *Market_id*)

```
SELECT    S.Product_Id, SUM (Sales_Amt)
FROM      Sales S, Time T
WHERE     T.Time_Id = S.Time_Id  AND  T.Week = 'Wk-12'
GROUP BY  S. Product_Id
```

Slice

Pivot

Slicing-and-Dicing

- Typically slicing and dicing involves several queries to find the "right slice."

  For instance, change the slice & the axes (from the prev. example):

  - Slicing on Time and Market dimensions then pivoting to *Product_id* and *Week* (in the time dimension)

```
SELECT      S.Product_Id, T.Quarter, SUM (Sales_Amt)
FROM        Sales S, Time T
WHERE       T.Time_Id = S.Time_Id
                    AND T.Quarter = 4
                    AND S.Market_id = 12345
GROUP BY    S.Product_Id, T.Week
```

Slice

Pivot

The CUBE Operator

- To construct the following table, would take 4 queries (next slide)

Market_Id

Product_Id

SUM(*Sales_Amt*)	M1	M2	M3	*Total*
P1	3003	1503	…	…
P2	6003	2402	…	…
P3	4503	3	…	…
P4	7503	7000	…	…
Total	…	…	…	…

The Three Queries

- For the table entries, without the totals (aggregation on time)

```
SELECT      S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM         Sales S
GROUP BY  S.Market_Id, S.Product_Id
```

- For the row totals (aggregation on time and markets)

```
SELECT      S.Product_Id, SUM (S.Sales_Amt)
FROM          Sales S
GROUP BY   S.Product_Id
```

- For the column totals (aggregation on time and products)

```
SELECT       S.Market_Id, SUM (S.Sales)
FROM          Sales S
GROUP BY    S.Market_Id
```

- For the grand total (aggregation on time, markets, and products)

```
SELECT       SUM (S.Sales)
FROM          Sales S
```

Definition of the CUBE Operator

- Doing these three queries is wasteful
  - The first does much of the work of the other two: if we could save that result and aggregate over *Market_Id* and *Product_Id*, we could compute the other queries more efficiently
- The CUBE clause is part of SQL:1999
  - GROUP BY CUBE (v1, v2, …, vn)
  - Equivalent to a collection of GROUP BYs, one for each of the 2^n subsets of v1, v2, …, vn

Example of CUBE Operator

- The following query returns all the information needed to make the previous products/markets table:

```
SELECT  S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM  Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```

ROLLUP

- ROLLUP is similar to CUBE except that instead of aggregating over all subsets of the arguments, it creates subsets moving from right to left
- GROUP BY ROLLUP (A_1,A_2,…,A_n) is a series of these aggregations:
  - GROUP BY A_1 ,…, A_{n-1} ,A_n
  - GROUP BY A_1 ,…, A_{n-1}
  - … … …
  - GROUP BY A_1, A_2
  - GROUP BY A_1
  - *No* GROUP BY
- ROLLUP is also in SQL:1999

Example of ROLLUP Operator

SELECT S.*Market_Id*, S.*Product_Id*, SUM (S.*Sales_Amt*)

FROM Sales S

GROUP BY ROLLUP (S.*Market_Id*, S. *Product_Id*)

- first aggregates with the finest granularity:

  GROUP BY S.*Market_Id*, S.*Product_Id*

- then with the next level of granularity:

  GROUP BY S.*Market_Id*

- then the grand total is computed with *no* GROUP BY clause

ROLLUP vs. CUBE

- The same query with CUBE:

  - first aggregates with the finest granularity:

  GROUP BY S.*Market_Id*, S.*Product_Id*

  - then with the next level of granularity:

  GROUP BY S.*Market_Id*

  and

  GROUP BY S.*Product_Id*

  - then the grand total with *no* GROUP BY

Materialized Views

The CUBE operator is often used to precompute aggregations on all dimensions of a fact table and then save them as a *materialized views* to speed up future queries

ROLAP and MOLAP

- Relational OLAP: ROLAP
  - OLAP data is stored in a relational database as previously described. Data cube is a conceptual view – way to *think about* a fact table
- Multidimensional OLAP: MOLAP
  - Vendor provides an OLAP server that *implements* a fact table as a data cube using a special multi-dimensional (non-relational) data structure

MOLAP

- No standard query language for MOLAP databases
- Many MOLAP vendors (and many ROLAP vendors) provide proprietary visual languages that allow casual users to make queries that involve pivots, drilling down, or rolling up

Implementation Issues

- OLAP applications are characterized by a very large amount of data that is relatively static, with infrequent updates
  - Thus, various aggregations can be precomputed and stored in the database
  - *Star joins*, *join indices*, and *bitmap indices* can be used to improve efficiency (recall the methods to compute star joins in Chapter 14)
  - Since updates are infrequent, the inefficiencies associated with updates are minimized

Data Warehouse

- Data (often derived from OLTP) for both OLAP and data mining applications is usually stored in a special database called a ***data warehouse***
- Data warehouses are generally large and contain data that has been gathered at different times from DBMSs provided by different vendors and with different schemas
- Populating such a data warehouse is not trivial

Issues Involved in Populating a Data Warehouse

- *Transformations*
  - *Syntactic*: syntax used in different DMBSs for the same data might be different
    - Attribute names: SSN vs. Ssnum
    - Attribute domains: Integer vs. String
  - *Semantic*: semantics might be different
    - Summarizing sales on a daily basis vs. summarizing sales on a monthly basis
- *Data Cleaning*
  - Removing errors and inconsistencies in data

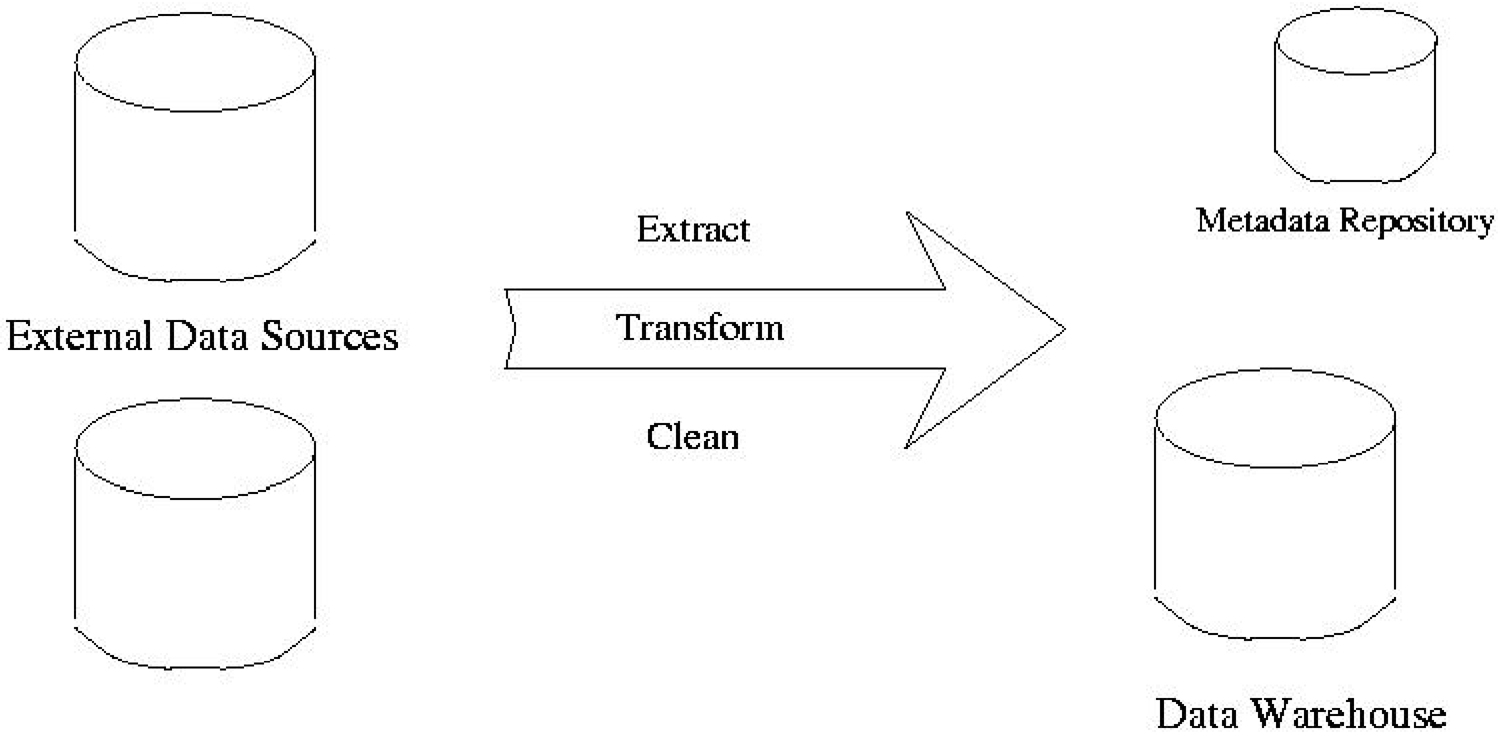
Metadata

- As with other databases, a warehouse must include a ***metadata repository***
  - Information about physical and logical organization of data
  - Information about the source of each data item and the dates on which it was loaded and refreshed

Incremental Updates

- The large volume of data in a data warehouse makes loading and updating a significant task
- For efficiency, updating is usually incremental
  - Different parts are updated at different times
- Incremental updates might result in the database being in an inconsistent state
  - Usually not important because queries involve only statistical summaries of data, which are not greatly affected by such inconsistencies

Loading Data into A Data Warehouse



Data Mining

- An attempt at knowledge discovery
- Searching for patterns and structure in a sea of data
- Uses techniques from many disciplines, such as statistical analysis and machine learning
  - These techniques are not our main interest

Goals of Data Mining

- **Association**
  - Finding patterns in data that associate instances of that data to related instances
    - Example: what types of books does a customer buy
- **Classification**
  - Finding patterns in data that can be used to classify that data (and possibly the people it describes)
    - Example "high-end buyers" and "low-end" buyers
  - This classification might then be used for **Prediction**
    - Which bank customers will default on their mortgages?
  - Categories for classification are known in advance

Goals (con’t)

- **Clustering**
  - Finding patterns in data that can be used to classify that data (and possibly the people it describes) into categories determined by a similarity measure
    - Example: Are cancer patients clustered in any geographic area (possibly around certain power plants)?
  - Categories are *not* known in advance, unlike is the classification problem

Associations

- An *association* is a correlation between certain values in a database (in the same or different columns)
  - *In a convenience store in the early evening, a large percentage of customers who bought diapers also bought beer*
- This association can be described using the notation

  Purchase_diapers => Purchase_beer

Confidence and Support

- To determine whether an association exists, the system computes the ***confidence*** and ***support*** for that association
- *Confidence* in A => B
  - The percentage of transactions (recorded in the database) that contain B among those that contain A
    - Diapers => Beer:

      The percentage of customers who bought beer among those who bought diapers
- *Support*
  - The percentage of transactions that contain both items among all transactions
    - 100* (customers who bought both Diapers and Beer)/(all customers)

Ascertain an Association

- To ascertain that an association exists, both the confidence and the support must be above a certain threshold
  - Confidence states that there is a high probability, given the data, that someone who purchased diapers also bought beer
  - Support states that the data shows a large percentage of people who purchased both diapers and beer (so that the confidence measure is not an accident)

A Priori Algorithm for Computing Associations

- Based on this observation:
  - If the support for A => B is larger than *T*, then the support for *A* and *B* must separately be larger than *T*
- Find all items whose support is larger than *T*
  - Requires checking *n* items
  - If there are *m* items with support > T (presumably, m<<n), find all pairs of such items whose support is larger than *T*
  - Requires checking *m*(*m*-*1*) pairs
- If there are *p* pairs with support > T, compute the confidence for each pair
  - Requires checking *p* pairs

Classification

- *Classification* involves finding patterns in data items that can be used to place those items in certain categories.That classification can then be used to predict future outcomes.
  - *A bank might gather data from the application forms of past customers who applied for a mortgage and classify them as* ***defaulters*** *or* ***non-defaulters****.*
  - *Then when new customers apply, they might use the information on their application forms to predict whether or not they would default*

Example: Loan Risk Evaluation

- Suppose the bank used only three types of information to do the classification
  - Whether or not the applicant was married
  - Whether or not the applicant had previously defaulted
  - The applicants current income
- The data about previous applicants might be stored in a table called the *training table*

Training Table

Id	***Married***	***PreviousDefault***	***Income***	***Default*** (outcome)
C1	Yes	No	50	No
C2	Yes	No	100	No
C3	No	Yes	135	Yes
C4	Yes	No	125	No
C5	Yes	No	50	No
C6	No	No	30	No
C7	Yes	Yes	10	No
C8	Yes	No	10	Yes
C9	Yes	No	75	No
C10	Yes	Yes	45	No

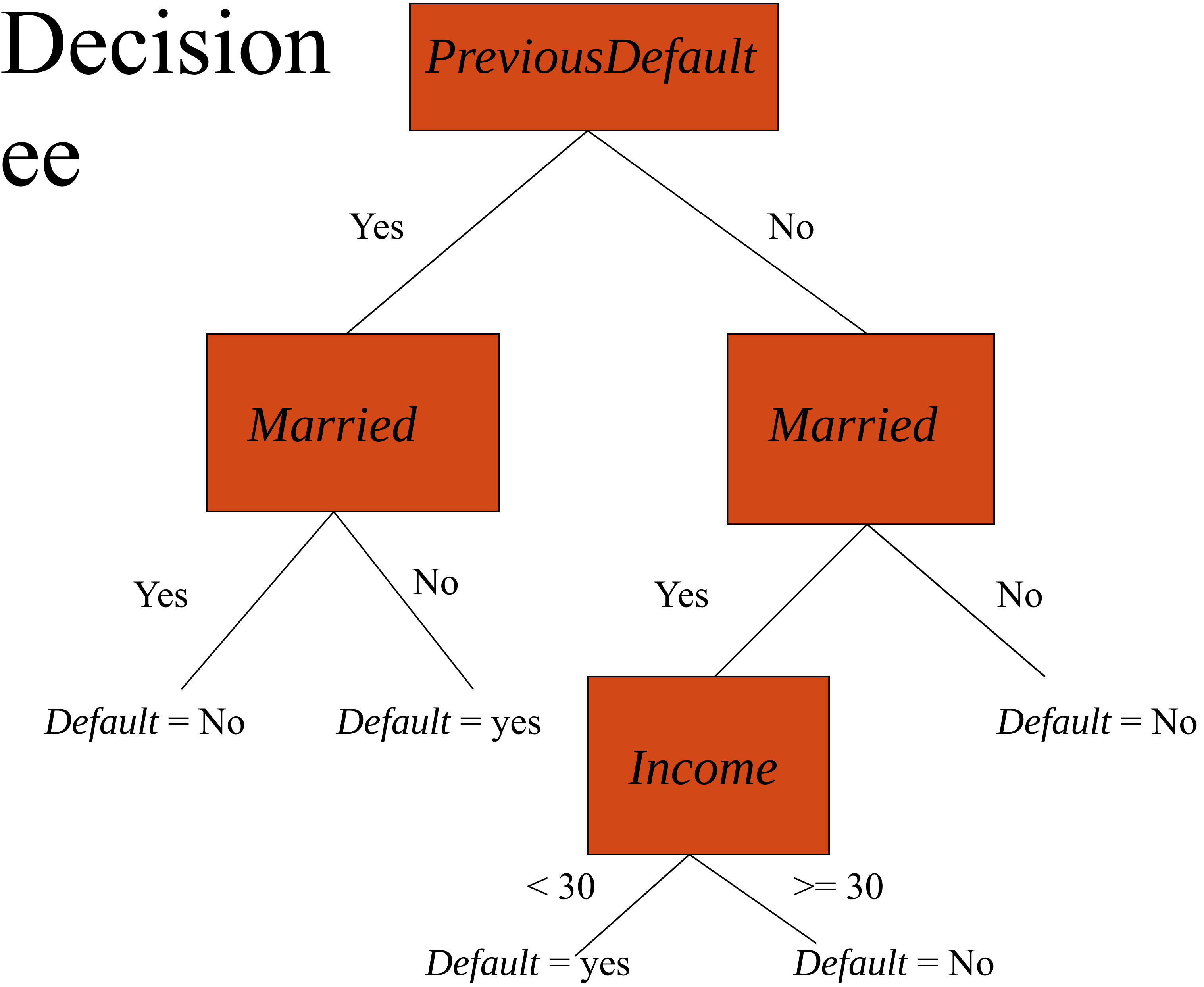
Training Table

Id	***Married***	***PreviousDefault***	***Income***	***Default*** (outcome)
C11	Yes	No	60	Yes
C12	No	Yes	125	Yes
C13	Yes	Yes	20	No
C14	No	No	15	No
C15	No	No	60	No
C16	Yes	No	15	Yes
C17	Yes	No	35	No
C18	No	Yes	160	Yes
C19	Yes	No	40	No
C20	Yes	No	30	No

Classification Using Decision Trees

- The goal is to use the information in this table to classify new applicants into defaulters or non defaulters
- One approach is to use the training table to make a decision tree

A Decision Tree



PreviousDefault

Yes

No

Married

Married

Yes

No

Yes

No

Default = No

Default = yes

Income

Default = No

< 30

>= 30

Default = yes

Default = No

Decision Trees Imply Classification Rules

- Each classification rule implied by the tree corresponds to a path from the root to a leaf
- For example, one such rule is

If $\quad PreviousDefault = \text{No} \quad \text{AND} \quad Married = \text{Yes} \quad \text{AND} \quad Income < 30$

Then $Default = \text{Yes}$

Decision Trees Might Make Mistakes

- Some of the classification rules developed from a decision tree might incorrectly classify some data; for example

  If $\quad \textit{PreviousDefault} = \text{No}\ \text{ AND }\ \textit{Married} = \text{Yes}\ \text{ AND }\ \textit{Income} >= 30$

  Then $\ \textit{Default} = \text{No}$

  does not correctly classify customer C11

- It is unreasonable to expect that a small number of classification rules can always correctly classify a large amount of data
  - Goal: Produce the best possible tree from the given data

Producing a Decision Tree From a Training Set

- Several algorithms have been developed for constructing a decision tree from a training set
  - We discuss the *ID3 algorithm*
- ID3 starts by selecting the attribute to be used at the top level of the tree to make the first decision
- This decision yields the nodes at the second level of the tree. The procedure repeats on each of these nodes

Picking the Top-Most Attribute

- Intuitively we want to pick the attribute that gives the "most information" about the final decision
- The ID3 algorithm uses the ***entropy*** measure from Information Theory

$$entropy(\text{TrainingTable}) = -\sum_{i \in \text{outcomes}} p_i \log_2 p_i$$

☞ p_i = probability of the outcome of *i* in TrainingTable

- Practically: p_i is approximated as

  p_i = (#*items in the table with outcome=i) / (# of all items in the table)*

Properties of the Entropy $-\ \Sigma\ p_i \log_2 p_i$

- Entropy determines the degree of randomness in the data:
  - $p_{yes} = p_{no} = ½$ – *data is completely random*

    $entropy = –\ ½ \log_2 ½ – ½ \log_2 ½ = ½ + ½ = 1$
  - $p_{yes}= 1, p_{no}= 0$ or $p_{no}= 1, p_{yes}= 0$ – *data is totally nonrandom*

    $entropy = –\ 1 \log_2 1 – 0 \log_2 0 = 0$
- The lower the entropy – the less randomness is in the data ≡ the more information is in the data

Information Gain

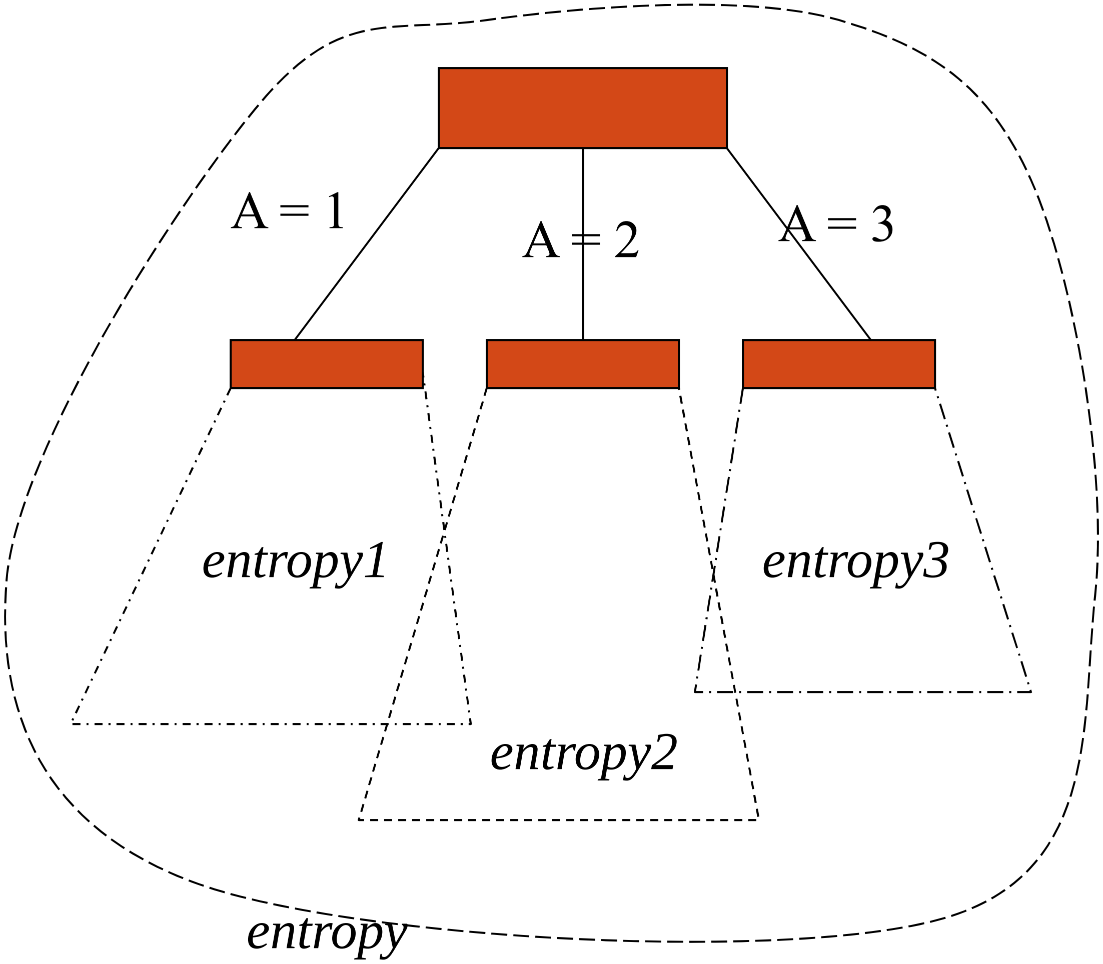
- For the entire table, 6 entries have the outcome “Yes”and 14 have the outcome “No”
  - So the entropy of the entire table is
    - $-(6/20 \log_2 6/20 + 14/20 \log_2 14/20) = .881$
- The ID3 algorithm selects as the top-most node the attribute that provides the largest ***Information Gain*** (explained next)

Information Gain (cont'd)

- Select an attribute, A, and compute the entropies of the subtrees w.r.t. that attribute
- **Information gain**:

  $entropy - (\Sigma_{i=1..n}\ entropy_i)/n$

  - How much less random the data has become after splitting the training set into subtrees



Information Gain (con’t)

- If the top-most node in the tree were *Previous Default*, there would be two subtrees:

  a subtree with *Previous Default* = “Yes”

  a subtree with *Previous Default* = “No”

- The entropies of these two subtrees would be
  - For *Previous Default* = “Yes”:
    - 4 of the 6 entries have the outcome “Yes” and 2 have “No”
      - The entropy is $-4/6 \log_2 4/6 - 2/6 \log_2 2/6 = .918$
  - For *Previous Default* = “No”:
    - 2 of the 14 entries have the outcome “Yes” and 12 have “No”
      - The entropy is $-2/14 \log_2 2/14 - 12/14 \log_2 12/14 = .592$
- The average entropy of these subtrees is $(.918+.592)/2 = .690$
- The Information Gain from using *Previous Default* as the top attribute is $.881 - .690 = .191$

Comparing Information Gains

- *Previous Default* as the top-most attribute
  - The information gain = .191
- *Married* as the top-most attribute
  - The information gain = .036
- *Income* as the top-most attribute
  - Must compute information gain for all possible ranges
  - For example for the range Income < 50 and Income >= 50 the Information Gain is .031
- The maximum Information Gain turns out to be for the attribute *Previous Default*, so we select that as the top-most attribute in the decision tree

The Rest of the Tree

- Repeat the process on the each of the subtrees
  - Different subtrees might have different top-most nodes and/or different ranges for *Income*
  - If all nodes in a subtree have the same outcome:
    - the subtree becomes a leaf node and the procedure stops for that subtree
  - If all nodes in a subtree do not have the same outcome:
    - If there are *no* more attributes to use: That subtree becomes a leaf node and the procedure stops for that subtree
      - *The classification rules that access such a subtree will incorrectly classify* <u>*some*</u> *data.*

      E.g., the subtree *PreviousDefault* = No AND *Married* =Yes AND *Income*>=30 incorrectly classifies C11.
    - If there *are* more attributes to use: Continue the process

Other Measures

- A number of other measures can be used to produce a decision tree from a training set
- Gain Ratio = (Information Gain)/SplitInfo
  - Where SplitInfo = $-\Sigma \mid t_i \mid / \mid t \mid * \log_2 \mid t_i \mid / \mid t \mid$
  - |t| is the number of entries in the table being decomposed and $\mid t_i \mid$ is the number of entries in the i^{th} table produced
- Gini Index = $1 - \Sigma\ p_i^2$

Neural Networks : Another Approach to Classification and Prediction

- Machine Learning
  - A mortgage broker believes that several factors might affect whether or not a customer is likely to default on mortgage, but does now know how to weight these factors
  - Use data from past customers to "learn" a set of weights to be used in the decision for future customers
    - Neural networks, a technique studied in the context of Artificial Intelligence, provides a model for analyzing this problem
    - Various learning algorithms have been proposed in the literature and are being used in practice

A Model of a Neuron

- Suppose the factors are represented as x_i where each x_i can be 1 or 0, and the weight of each such factor is represented as w_i Then the weighted sum of the factors is compared with a threshold t. If the weighted sum exceeds the threshold

$$\sum_{i=1}^{n} w_i \times x_i \geq t$$

the output is 1 and we predict that the customer will default; otherwise the output is 0 and we predict he would be considered a good risk

Simplified Model

- The model is simplified if we introduce a new weight w_0, which equals t, and assume there is a new input x_0 which always equals -1. Then the above inequality becomes

$$\sum_{i=0}^{n} w_i \times x_i \geq 0$$

Step-Function Activation

- This model is said to have **step-function activation**
  - Its output is 1 if the weighted sum of the inputs is greater than or equal to 0
  - Its output is 0 otherwise
- Neurons with this activation function are sometimes called **perceptrons**.
- Later we will discuss another activation function

Perceptron Learning Algorithm

- Set the values of each weight (and threshold) to some small random number
- Apply the inputs one at a time and compute the outputs
- If the desired output for some input is d and the actual output is y, change each weight w_i by

$$\Delta w_i = \eta \times x_i \times (d - y)$$

where η is a small constant called the **learning factor**

- Continue until some termination condition is met

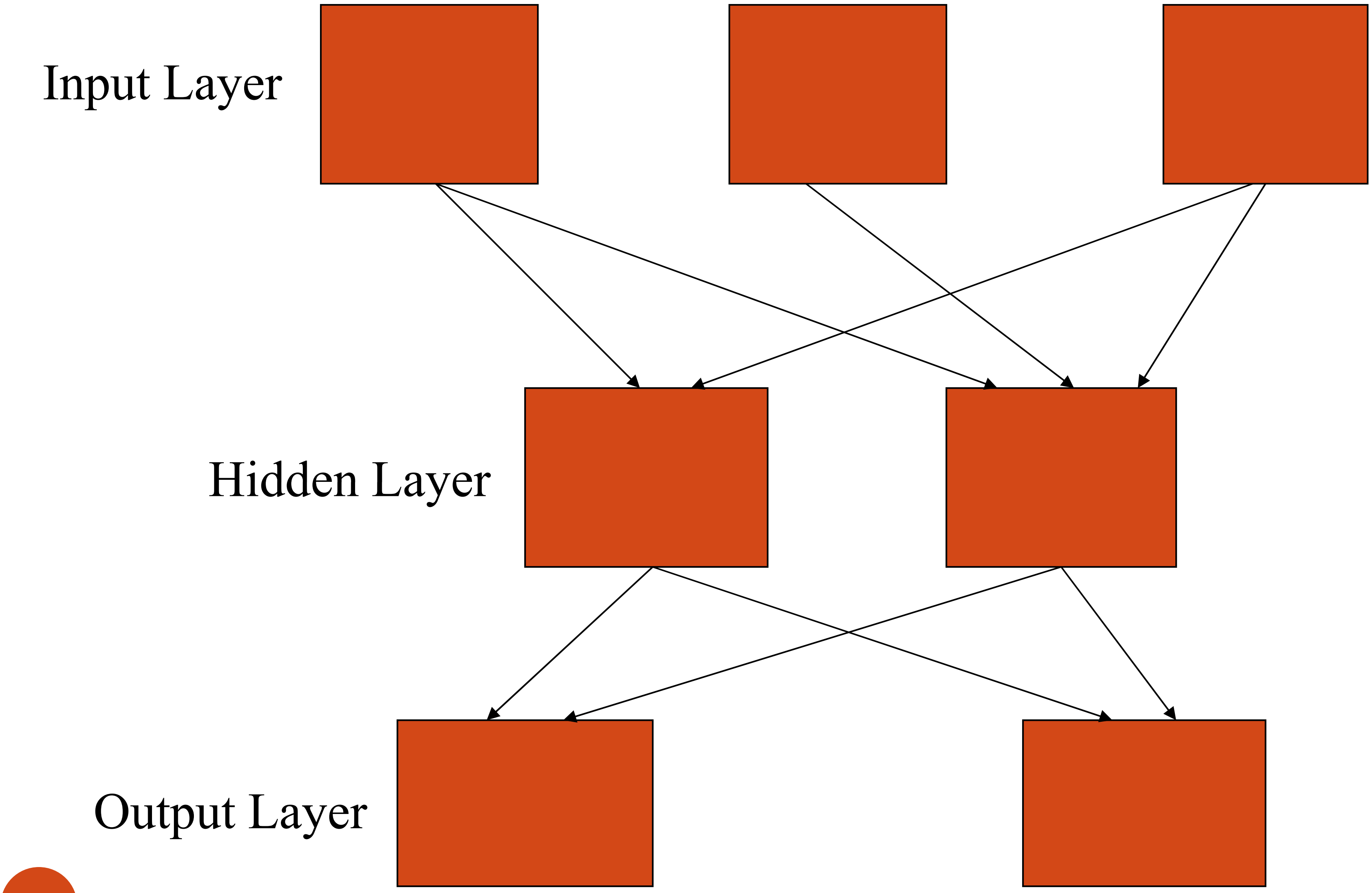
Rationale for Learning Algorithm

- If there is no error, no change in the weights are made
- If there is an error, each weight is changed in the direction to decrease the error
  - For example if the output is 0 and the desired output is 1, the weights of all the inputs that were 1 are increased and the threshold is decreased.

Correctness and Problems with Perceptron Learning Algorithm

- If the decision can always be made correctly by a single neuron, this algorithm will eventually "learn" the correct weights
- The problem is that, for most applications, the decision cannot be made, even approximately, by a single neuron
- We therefore consider networks of such neurons

Three Level Neural Network



Input Layer

Hidden Layer

Output Layer

Three-Level Network

- The input level just gathers the inputs and submits them to the other levels (no neurons)
- The middle or hidden level consists of neurons that make intermediate decisions and send them to the output layer
- The output layer makes the final decisions

The Sigmoid Activation Function

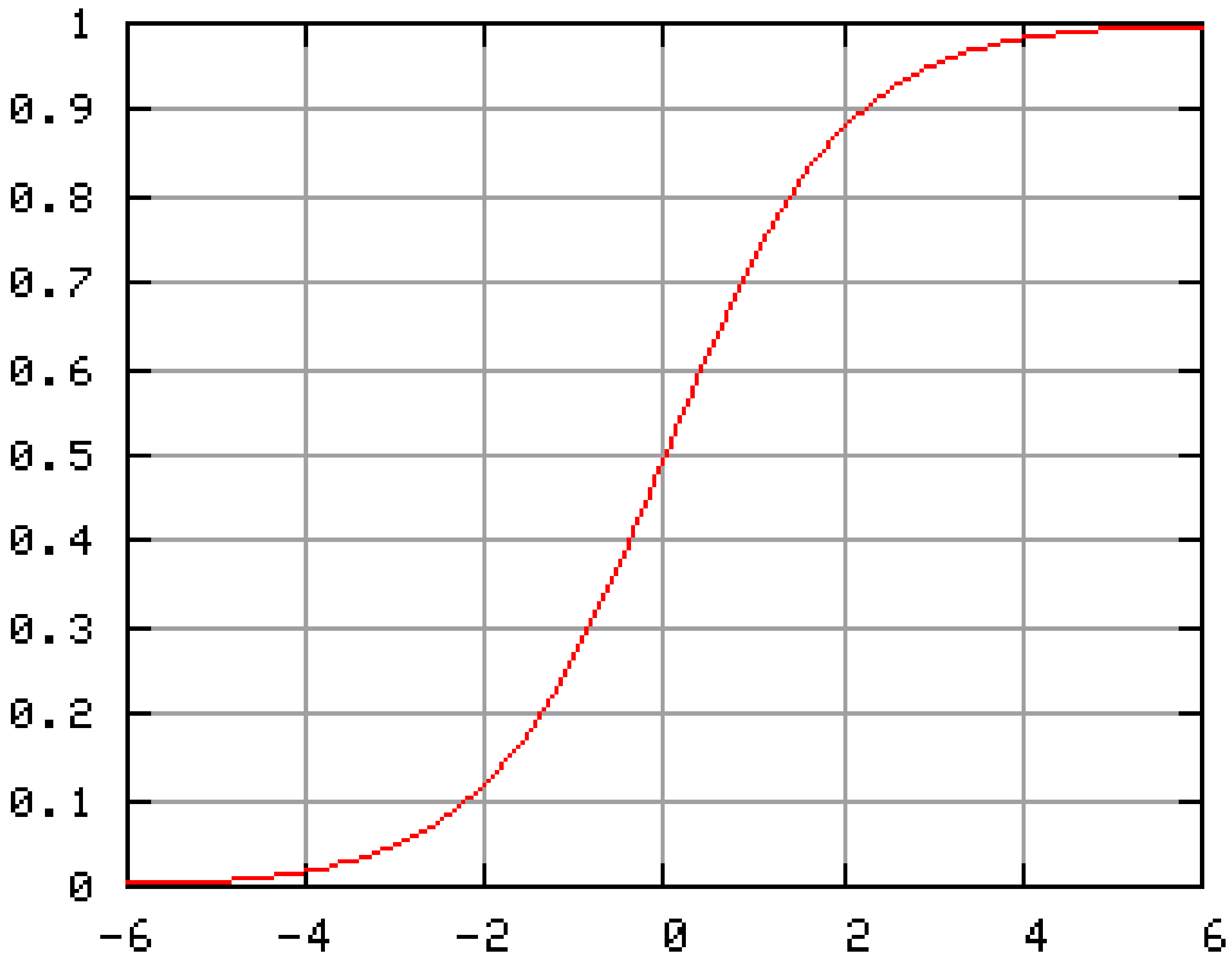
- To mathematically derive a learning algorithm for such a neural network, we must take derivatives
  - But we cannot take derivatives of the step function activation function
- Therefore we must use a continuous activation function
  - A common such activation function is the sigmoid function

$$y = 1/(1+e^{-X})$$

where

$$X = \sum_{i=0}^{n} w_i \times x_i$$

The Sigmoid Function



Properties of Sigmoid Function

- In some sense the sigmoid function is similar to the step function
  - It has the value *.5* for $x = 0$
  - It becomes asymptotic to *1* for large positive values of x
  - It becomes asymptotic to *0* for large negative values of x
- However it is continuous and, as can be easily computed, has the derivative

$$\frac{\partial y}{\partial X} = e^{-X} / (1 + e^{-X})^2 = y \times (1 - y)$$

which is used in many of the following computations

Learning Algorithm for a Single Sigmoid Neuron

- The idea is to take the derivative of the squared error with respect to each of the weights and change each weight by a small multiple of the negative of that derivative
  - Called the Gradient Descent Approach
  - Move in the direction towards the minimum of the function

$$\Delta w_i = -\eta \times \frac{\partial (d - y)^2}{\partial w_i}$$

The Algorithm for One Neuron (continued)

- After a bit of math, and using the previous result for the derivative of the sigmoid function, we get

$$\Delta w_i = \eta \times x_i \times y \times (1 - y) \times (d - y)$$

Back Propagation Algorithm for 3-Level Neural Network

- Initially set the values of all weights to some small random number
- Apply the inputs from the learning set one at a time and, for each input, compute the outputs of the neurons in the output layer

Back Propagation Algorithm (continued)

- Adjust the weights of each neuron in the outer layer
- Using the same reasoning as for the single neuron

$$\Delta w_i^{out} = \eta \times x_i^{out} \times y^{out} \times (1 - y^{out}) \times (d^{out} - y^{out})$$

Back Propagation Algorithm (continued)

- For reasons that will be clear later, this equation can be simplified to

$$\Delta w_i^{out} = \eta \times x_i^{out} \times \delta_i^{out}$$

where

$$\delta_i^{out} = y^{out} \times (1 - y^{out}) \times (d^{out} - y^{out})$$

Back Propagation Algorithm (continued)

- Now consider neurons in the hidden layer. Assume first that there is only one neuron in the output layer
- Using the same reasoning as before, the gradient descent method tells us that

$$\Delta w_i{}^{mid} = -\eta \times \frac{\partial (d^{out} - y^{out})^2}{\partial w_i{}^{mid}}$$

Back Propagation Algorithm (continued)

- Doing the math, we get

$$\Delta w_i^{mid} = \eta \times x_i^{mid} \times \delta^{mid}$$

where

$$\delta^{mid} = y^{mid} \times (1 - y^{mid}) \times w^{mid/out} \times \delta^{out}$$

and where δ^{out} was previously computed (the back propagation property)

Back Propagation Algorithm (continued)

- If there is more than one neuron in the output layer, we compute

$$\Delta w_i^{mid} = -\eta \times \frac{\partial \sum_j (d_j^{out} - y_j^{out})^2}{\partial w_i^{mid}}$$

$$= \eta \times x_i^{mid} \times \delta^{mid}$$

where

$$\delta^{mid} = y^{mid} \times (1 - y^{mid}) \times \sum_j (w_j^{mid/out} \times \delta_j^{out})$$

Back Propagation Algorithm (continued)

- Continue the training until some termination condition is met
  - The data in the training set has been used some fixed number of times
  - The number of errors has stopped decreasing significantly
  - The weights have stopped changing significantly
  - The number of errors has reached some predetermined level

Clustering

- Given:
  - a set of items
  - characteristic attributes for the items
  - a similarity measure based on those attributes
- ***Clustering*** involves placing those items into ***clusters***, such that items in the same cluster are close according to the similarity measure
  - Different from Classification: there the categories are known in advance
- For example, cancer patients might have the attribute *location*, and might be placed in clusters with similar locations.

Example: Clustering Students by Age

Student Id	**Age**	**GPA**
S1	17	3.9
S2	17	3.5
S3	18	3.1
S4	20	3.0
S5	23	3.5
S6	26	2.6

K-Means Algorithm

- To cluster a set of items into *k* categories
  1. Pick *k* items at random to be the (initial) centers of the clusters (so each selected item is in its own cluster)
  2. Place each item *in the training set* in the cluster to which it is closest to the center
  3. Recalculate the centers of each cluster as the mean of the items in that cluster
  4. Repeat the procedure starting at Step 2 until there is no change in the membership of any cluster

The Student Example (con't)

- Suppose we want 2 clusters based on *Age*
  - Randomly pick S1 (age 17) and S4 (age 20) as the centers of the initial centers
  - The initial clusters are

    17 17 18 20 23 26
  - The centers of these clusters are

    17.333 and 23
  - Redistribute items among the clusters based on the new centers:

    17 17 18 20 23 26
  - If we repeat the procedure, the clusters remain the same

The Hierarchical or Agglomerative Algorithm

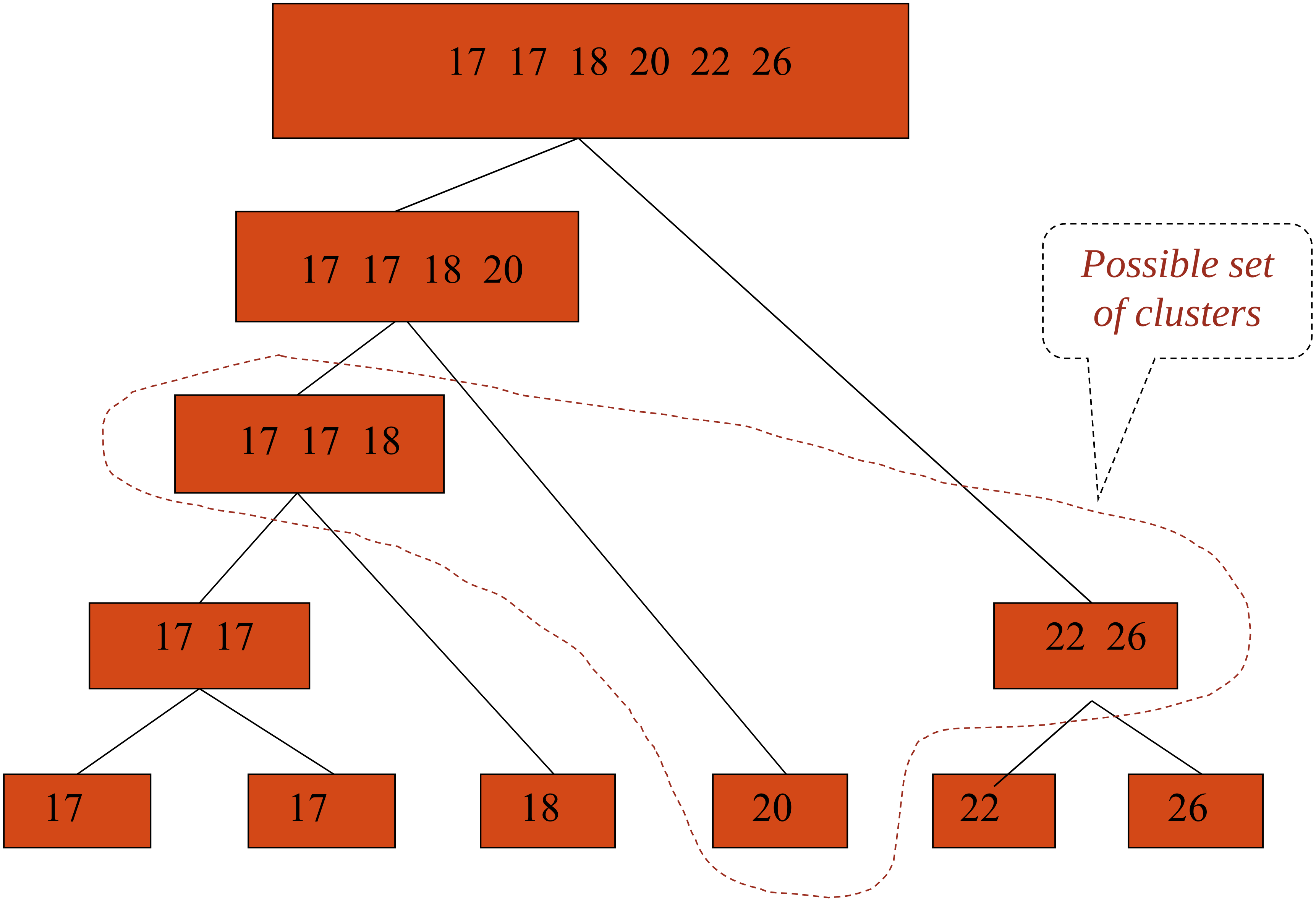
- Number of clusters is not fixed in advance
- Initially select each item in the training set as the center of its own cluster
- Select two clusters to merge into a single center
    - One approach it to pick the clusters whose centers are closest according to some measure (e.g., Euclidian distance)
- Continue until some termination condition is reached (e.g., the number of clusters falls below some limit)

Student Example (con't)

```
17    17   18   20   23   26
17 17      18   20   23   26
17 17 18        20   23   26
17 17 18 20          23   26
17 17 18 20          23 26 --- K-means Solution
17 17 18 20 23 26
```

Dendrogram

- One way to manually *analyze* the results of the hierarchical algorithm is with the use of a tree called a ***dendrogram***
- The nodes are clusters in the intermediate stages of the hierarchical algorithm
- The tree is constructed in reverse order of the execution of the hierarchical algorithm, starting with the final (single) cluster



A Dendrogram for the Student Example

Analysis of Dendrogram

- Any set of nodes whose children *partition* all the leaves is a possible clustering
  - For example

17 17 18   20   23 26

is an allowable set of clusters.

Note: these clusters were not seen at any of

the intermediate steps in the hierarchical

or K- means algorithms!