Introduction to Data Mining and Machine Learning

Redwan Ahmed Rizvee
https:/rizveeredwan.github.io/
YouTube Lecture Link

Database Applications: **Transaction Processing** vs. **Decision Support**

1. Transaction-Processing Systems (TPS)

- **Purpose:** Record detailed information about **transactions** (e.g., sales, registrations, grades).
- **Characteristics:** Widely used, accumulate vast amounts of detailed data (gigabytes to terabytes).
- **Examples:** Product sales data, course registration, customer and transaction information (items purchased, price, date, customer details).

2. Decision-Support Systems (DSS)

- **Purpose: Extract high-level insights** from TPS data to aid decision-making.
- **Goal:** Help managers **make strategic decisions** (e.g., product stocking, manufacturing, university admissions, marketing strategies).
- **Example Applications:**
  - Identifying sales trends (e.g., flannel shirts in Pacific Northwest). => Time series analysis
  - Targeting marketing based on customer demographics (e.g., small sports cars for young women with specific income). => Data Mining, Machine Learning
- **Challenge: Storing** and **retrieving** data for decision support presents several issues.

Challenges and Solutions in Decision Support

- **SQL Limitations:**
  - Many DSS queries are complex or cannot be easily expressed in standard SQL.
  - **Solution:** SQL extensions for data analysis (e.g., OLAP techniques).
- **Statistical Analysis:**
  - Database query languages are not ideal for **detailed statistical analysis**.
  - **Solution:** Interfacing databases with specialized statistical packages (e.g., SAS, S++).
- **Diverse Data Sources:**
  - Large companies have data from multiple sources with different schemas, often not easily retrievable on demand due to performance/control issues.
  - **Solution: Data Warehouses** – consolidate data from multiple sources into a unified schema at a single site, providing a uniform interface.
- **Knowledge Discovery & Pattern Recognition:**
  - Need to **automatically discover statistical rules and patterns** from data.
  - **Solution: Data Mining** – combines AI and statistical techniques with efficient implementations for large databases to identify patterns in customer behavior and other data.

Data Warehousing: Addressing Data Challenges

1. Challenges with Diverse Data Sources for Decision Making

- **Geographical Distribution:** Data spread across many stores (e.g., retail chains) or branches (e.g., insurance companies).
- **Organizational Complexity:** Different data types (e.g., manufacturing problems, customer complaints) stored in various locations, operational systems, or under different schemas within a large organization.
- **External Data Integration:** Need to incorporate data purchased from outside sources (e.g., mailing lists, credit scores).
- **Cumbersome Querying:** Setting up queries across individual, disparate sources is inefficient and difficult.
- **Historical Data Access:** Operational systems often store only current data, but decision-makers require access to past data (e.g., changes in purchase patterns over time).

2. Data Warehouses: The Solution

- **Definition:** A data warehouse **is a centralized repository** (archive) of information.
- **Key Characteristics:**
  - **Multiple Sources:** Gathers data from diverse internal and external sources.
  - **Unified Schema:** Stores all collected data under a **single, consistent schema**.
  - **Single Site:** Consolidates data at **one location**.
  - **Historical Data:** Data is stored for a long time, enabling access to historical trends and past information.

3. Benefits of Data Warehouses

- **Consolidated Interface:** Provides **users with a single, uniform interface** to all data, simplifying decision-support queries.
- **Improved Efficiency:** Makes it easier and more efficient to write complex decision-support queries.
- **Reduced Impact on TPS:** Separates decision-support workloads from online transaction-processing systems, ensuring TPS performance is not affected.

Key Issues & ETL Tasks in Building a Data Warehouse

The challenge of View Maintenance over the collected data from different sources

- **When and How to Gather Data:**
  - **Source-Driven:** Sources transmit new data.
  - **Destination-Driven:** Warehouse requests new data.
  - **Consistency:** Typically slightly out-of-date (acceptable for DSS).
- **What Schema to Use:**
  - **Challenge:** Different schemas/data models at sources.
  - **Solution:** Schema integration; data is a **materialized view** of sources.
- **Data Transformation and Cleansing:**
  - **Cleansing:** Correcting inconsistencies (e.g., misspelled names/addresses, deduplication, householding).
  - **Transformation:** Changing units, converting schemas.
  - **Tools:** Graphical tools often support data flow specification.
- **How to Propagate Updates:**
  - **Challenge:** Reflecting source updates in the warehouse.
  - **Solution: View-maintenance problem**.
- **What Data to Summarize:**
  - **Challenge:** Raw data can be too large.
  - **Solution:** Maintain summary data (aggregations) for efficient querying.
- **ETL Tasks (Extract, Transform, Load):**
  - **Extract:** Getting data from sources.
  - **Transform:** Cleansing and transforming data.
  - **Load:** Loading processed data into the warehouse.

Data Warehouse Architecture

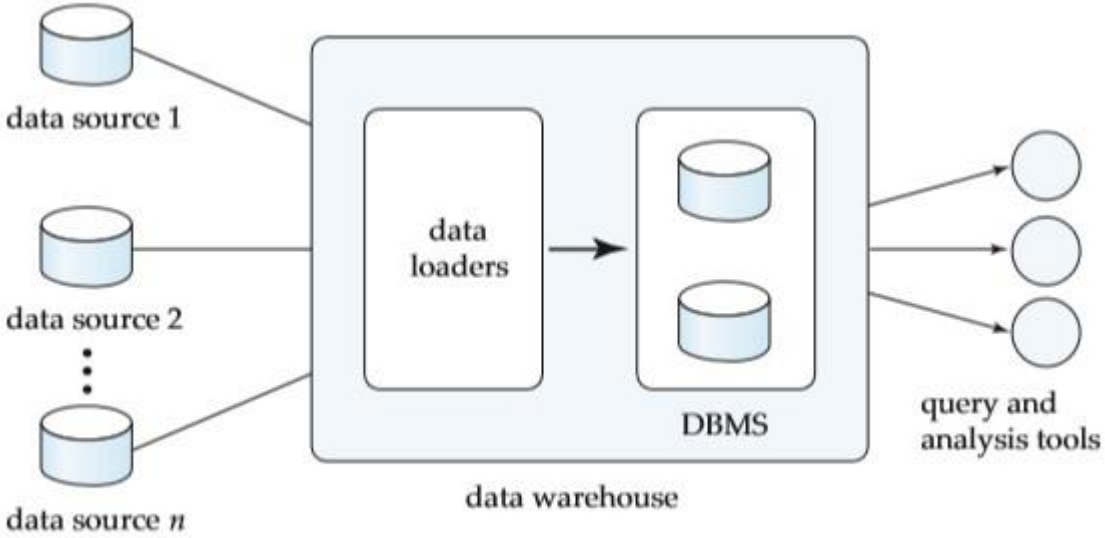
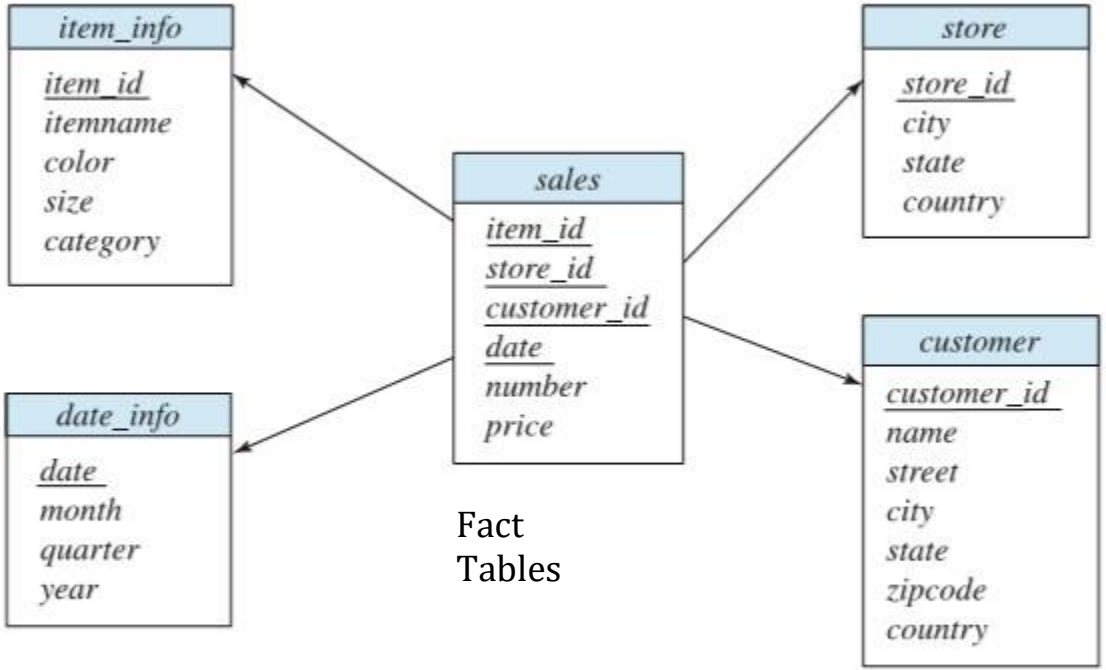


Figure 20.1 Data-warehouse architecture.

Schema Design for Data Warehouse

Dimension Tables & Attributes



Fact Tables

Figure 20.2 Star schema for a data warehouse.

Designed for Data Analysis

- Data warehouse schemas are optimized for analysis, often with *OLAP* tools, using **multidimensional** data with **dimension** and **measure attributes**.

Fact Tables

- **Fact tables** store multidimensional data, are typically very large, and contain **measure attributes** (quantitative values like items sold, price) that can be aggregated. Example: A sales table with one tuple per item sold.

Dimension Attributes

- **Dimension attributes** provide context for measures and are typically short identifiers serving as foreign keys. For a sales table, examples include Item ID, Date, Store ID, and Customer ID.

Dimension Tables

- **Dimension tables hold detailed information** for each dimension attribute, **minimizing fact table storage**. Fact tables link to them via foreign keys.

Column-Oriented Storage

- **Row-Oriented Storage (Traditional):** Stores all attributes of a tuple together, sequentially in a file.
- **Column-Oriented Storage (Contrast):** Stores each attribute of a relation in a separate file; values from successive tuples are stored at successive positions.

Benefits of Column-Oriented Storage:

1. **Efficient Querying:** When a query needs only a few attributes, only those relevant attributes are fetched, saving memory and cache space.
2. **Improved Compression:** Storing similar data types together enhances compression effectiveness, reducing disk storage and retrieval time.

Drawbacks:

- Fetching or storing a single tuple requires multiple I/O operations.

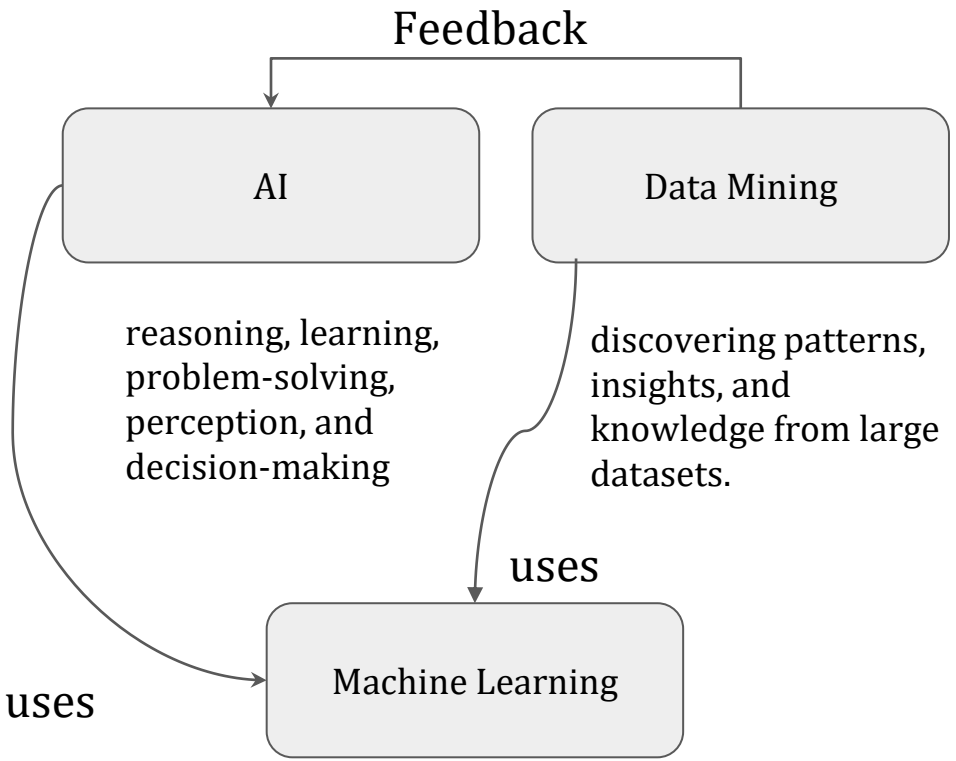
Application in Data Warehousing:

- Not widely used for transaction-processing (due to single-tuple I/O drawback).
- **Gaining acceptance for data warehousing:** Ideal for applications that involve scanning and aggregating multiple tuples (common in DSS queries), rather than accessing individual tuples.
- Demonstrates significant performance gains for many data-warehousing applications.

Data Mining

Data Mining

- **Definition:** The semiautomatic process of **analyzing large databases** to find **useful patterns**, often referred to as "knowledge discovery in databases."
- **Distinction:** Differs from machine learning and statistics **by focusing on large volumes of data primarily stored on disk**.
- **Process:** Typically involves a semi automatic approach, including manual preprocessing of data and post-processing of discovered patterns to identify useful ones.



allow computers to *learn from data* and make predictions or decisions without being explicitly programmed for every scenario

Classification (A Type of Data Mining Prediction)

- **Problem:** Given items belonging to **several known classes** and **past training instances**, predict the class a new item belongs to, using its other attributes.
- **Method:** Often involves **finding rules that partition data** into disjoint groups.
- **Example (Credit Card Application):**
  - A company wants to predict an applicant's credit-worthiness (excellent, good, average, bad).
  - It uses existing customer data (training set) with known creditworthiness based on payment history.
  - Rules are derived from attributes like education level and income (e.g., "P.degree = masters and P.income > 75,000 P.credit = excellent").
  - These rules are then applied to new applicants whose payment history is unknown.

$$\forall person\ P,\ P.degree = masters \textbf{ and } P.income > 75,000 \Rightarrow P.credit = excellent$$

$$\forall\ person\ P,\ P.degree = bachelors \textbf{ or } (P.income \geq 25,000 \textbf{ and } P.income \leq 75,000) \Rightarrow P.credit = good$$

Decision Tree Classifier:

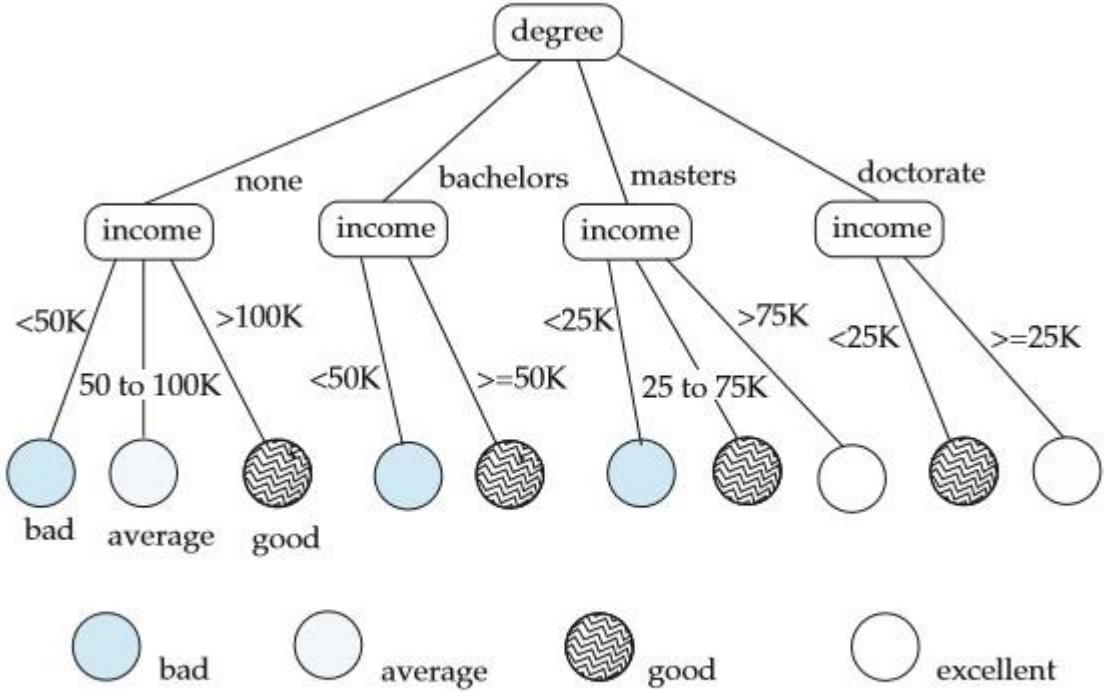


Figure 20.3 Classification tree.

Building Decision-Tree Classifiers

- **Method:** Most commonly built using a **greedy, recursive algorithm** that starts at the root and builds the tree downward.
- **Initial State:** *All training instances are associated with the root node*.
- **Node Processing:**
  - If "almost all" training instances at a node belong to the same class, the node becomes a **leaf node** associated with that class.
  - Otherwise, a **partitioning attribute** and **partitioning conditions** are selected to create child nodes.
  - Data associated with each child node consists of instances satisfying its partitioning condition.
- **Example (Credit Card Application):**
  - **First Split:** Attribute degree is chosen, creating four child nodes (e.g., degree = none, degree = bachelors, degree = masters, degree = doctorate).
  - **Second Split (under degree = masters):** Attribute income is chosen, with ranges like "0 to 25K," "25K to 50K," "50K to 75K," and "over 75K."
  - **Optimization:** If multiple ranges lead to the same class, they can be merged (e.g., "25K to 75K" for the "good" credit class).

Best Splits for Decision Trees

- **Goal:** To measure the "purity" of data at nodes and select the attribute and condition that result in the maximum purity at the children.
- **Purity Measures:**
  - **Gini Measure:** $\text{Gini}(S)=1-\sum_{i=1}^{k} p_i^2$ (where $\mathbf{p_i}$ is the fraction of instances in class i). *Gini is 0 for a single class and maximum when classes are equally distributed.*
  - **Entropy Measure:** $\text{Entropy}(S)=-\sum_{i=1}^{k} p_i \log_2(p_i)$. *Entropy is 0 for a single class* and *maximum when classes are equally distributed*.

Table 8.1 Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

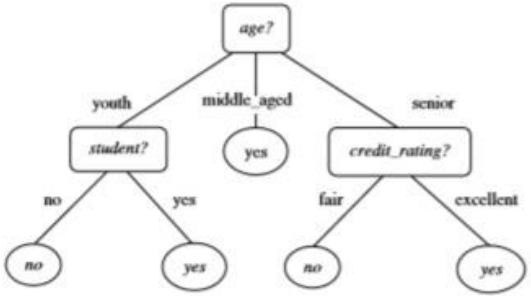


Figure 8.2 A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

$Gini(S) = 1 - (9/14)^2 - (5/14)^2 = 0.46$

$Entropy\ (S) = - (9/14) * \log(9/14) - (5/14) * \log(5/14) = 0.93$

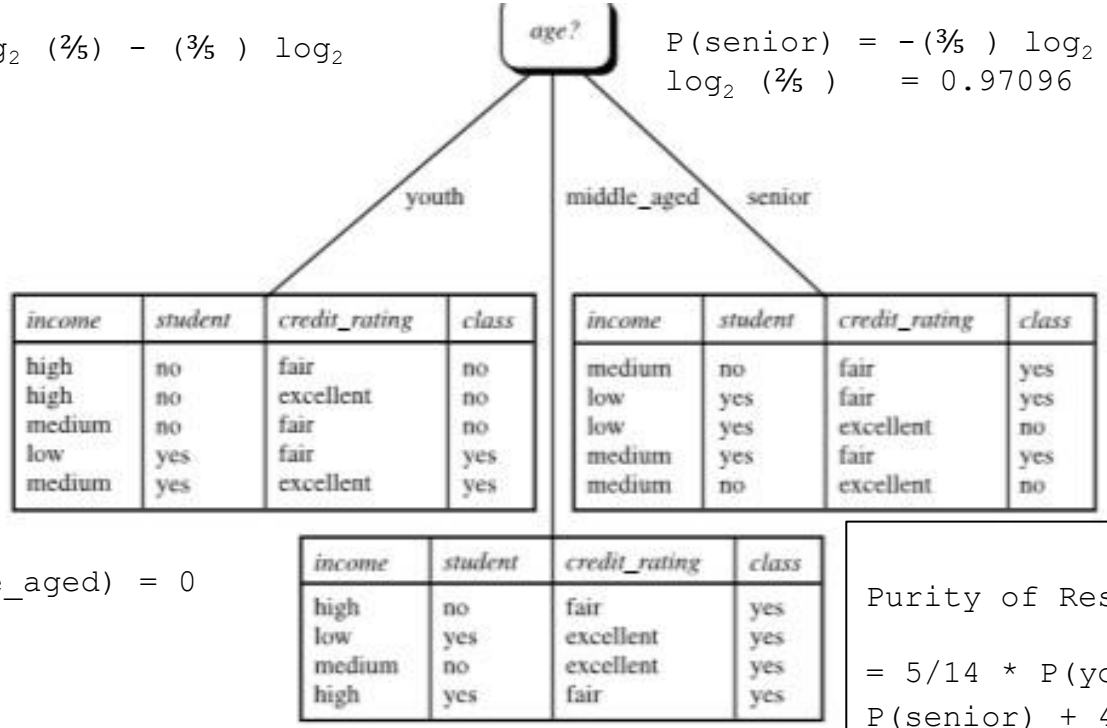
- **Purity of Resultant Sets:** For a split of set S into $S_1,S_2,\ldots,S_r$: Purity$(S_1,S_2,\ldots,S_r)=\sum_{i=1}^{r}(|S_i|/|S|)$ x purity(S_i) (weighted average).
- **Information Gain:** Measures the reduction in entropy (or Gini impurity) after a split: Information.gain $(S,\{S_1,S_2,\ldots,S_r\})$ = purity(S) − purity$(S_1,S_2,\ldots,S_r)$
- **Information Content of a Split:** Defined in terms of entropy: Information.content$(S,\{S_1,S_2,\ldots,S_r\})=-\sum_{i=1}^{r}(|S_i|/|S|)\log_2(|S_i|/|S|)$
- **Best Split Criterion:** The best split for an attribute is the one that gives the **maximum information gain ratio**, defined as:

$$\frac{\text{Information\_gain}(S,\{S_1,S_2,\ldots,S_r\})}{\text{Information\_content}(S,\{S_1,S_2,\ldots,S_r\})}$$

- **Preference for Fewer Splits:** Splits into fewer sets are generally preferred for simpler and more meaningful decision trees.

P(youth) = -(2/5) $\log_2$ (2/5) - (3/5) $\log_2$ (3/5) = 0.97096

P(senior) = -(3/5) $\log_2$ (3/5) - (2/5) $\log_2$ (2/5) = 0.97096



income	student	credit_rating	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

P(middle_aged) = 0

income	student	credit_rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Purity of Resultant Sets

= 5/14 * P(youth) + 5/14 * P(senior) + 4/14 * P(middle_aged) = (5/14) * 2 * 0.97096 = 0.694

Information Gain = 0.93 - 0.694 = 0.236

Information Content = -5/14 * $\log_2$(5/14) - 5/14 * $\log_2$(5/14) - 0 = 1.061

Gain Ratio = 0.236/1.061 = 0.22

Decision-Tree Construction Algorithm

```
procedure GrowTree(S)
    Partition(S);

procedure Partition (S)
    if (purity(S) > δp or |S| < δs ) then
        return;
    for each attribute A
        evaluate splits on attribute A;
    Use best split found (across all attributes) to partition
            S into S1, S2, . . . , Sr;
    for i = 1, 2, . . . , r
        Partition(Si);
```

Figure 20.4 Recursive construction of a decision tree.

Several of the algorithms also prune subtrees of the generated decision tree to reduce **overfitting**: A subtree is overfitted if it has been so highly tuned to the specifics of the training data that it makes many classification errors on other data

Association Rules

- **Definition:** Rules that describe relationships between items in large databases, often in the form $i_1,i_2,\ldots,i_n \Rightarrow i_0$. *Left side Antecedent, Right Side Precedent*.

Key Measures for Association Rules:

- **Support:**
  - A measure of what fraction of the population satisfies both the antecedent (items on the left side of) and the consequent (item on the right side of ⇒) of the rule.
  - Indicates how frequently the itemset (antecedent + consequent) appears in the dataset.
  - Example: If 0.001% of all purchases include milk and screwdrivers, the support for milk ⇒ screwdrivers is low. Rules with low support are often not statistically significant or interesting.
- **Confidence:**
  - A measure of how often the consequent is true when the antecedent is true.
  - Indicates the reliability of the rule.
  - Example: If bread ⇒ milk has a confidence of 80%, it means that in 80% of purchases where bread is bought, milk is also bought. High confidence (ideally close to 100% in business applications) is desirable.

Large Itemsets

- **Definition:** Sets of items that occur together with sufficient "support" (i.e., frequently enough) in the database.
- **Purpose:** Rules with high confidence are generated only from large itemsets.

Apriori Technique for Generating Large Itemsets

- **Approach:** A greedy algorithm that works in multiple passes over the data.
- **Pass 1:** Counts only single items. Sets with insufficient support are eliminated.
- **Subsequent Passes (e.g., Pass 2):** Considers sets with two items, and so on.
- **Optimization:** At the end of each pass, any set with insufficient support is eliminated, and none of its supersets need to be considered in further passes (because if a set is not frequent, any set containing it also cannot be frequent).
- **Termination:** Computation terminates when no set of a given size has sufficient support, meaning no larger sets can be frequent.