### Data Visualization

- Categorization of visualization methods:
  - Pixel-oriented visualization techniques
  - Geometric projection visualization techniques
  - Icon-based visualization techniques
  - Hierarchical visualization techniques
  - Visualizing complex data and relations

**Geometric:** Direct visualization; scatterplot and scatterplot matrices; landscape; parallel coordinates
**Icon-based:** chernoff faces; stick figures
**Hierarchical:** dimensional stack; tree-map; cone trees; infocube

## Distance on Numeric Data: Minkowski Distance

- Minkowski distance: A popular distance measure

$$d(i,j) = \sqrt[p]{|x_{i1}-x_{j1}|^p + |x_{i2}-x_{j2}|^p + \cdots + |x_{ij}-x_{jl}|^p}$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{il})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jl})$ are two $l$-dimensional data objects, and $p$ is the order (the distance so defined is also called $L$-$p$ norm)

- $p$ = 1: ($L_1$ norm) Manhattan (or city block) distance
  - E.g., the Hamming distance: the number of bits that are different between two binary vectors $d(i,j) = |x_{i1}-x_{j1}| + |x_{i2}-x_{j2}| + \cdots + |x_{il}-x_{jl}|$
- $p$ = 2: ($L_2$ norm) Euclidean distance

$$d(i,j) = \sqrt{|x_{i1}-x_{j1}|^2 + |x_{i2}-x_{j2}|^2 + \cdots + |x_{il}-x_{jl}|^2}$$

- $p \to \infty$: ($L_{max}$ norm, $L_\infty$ norm) "supremum" distance
  - The maximum difference between any component (attribute) of the vectors

$$d(i,j) = \lim_{p\to\infty} \sqrt[p]{|x_{i1}-x_{j1}|^p + |x_{i2}-x_{j2}|^p + \cdots + |x_{il}-x_{jl}|^p} = \max_{f=1}^{l} |x_{if}-x_{jf}|$$

## Proximity Measure for Binary Attributes

- A contingency table for binary data

| Object *i* \ Object *j* | 1 | 0 | sum |
|---|---|---|---|
| 1 | *q* | *r* | *q+r* |
| 0 | *s* | *t* | *s+t* |
| sum | *q+s* | *r+t* | *p* |

- Distance measure for symmetric binary variables $d(i,j) = \frac{r+s}{q+r+s+t}$
- Distance measure for asymmetric binary variables: $d(i,j) = \frac{r+s}{q+r+s}$
- Jaccard coefficient (*similarity* measure for *asymmetric* binary variables): $sim_{Jaccard}(i,j) = \frac{q}{q+r+s}$
- Note: Jaccard coefficient is the same as (a concept discussed in Pattern Discovery) $coherence(i,j) = \frac{sup(i,j)}{sup(i)+sup(j)-sup(i,j)} = \frac{q}{(q+r)+(q+s)-q}$

## Proximity Measure for Categorical Attributes

- Categorical data, also called nominal attributes
  - Example: Color (red, yellow, blue, green), profession, etc.
- Method 1: Simple matching
  - *m*: # of matches, *p*: total # of variables

$$d(i,j) = \frac{p-m}{p}$$

- Method 2: Use a large number of binary attributes
  - Creating a new binary attribute for each of the *M* nominal states

## Ordinal Variables

- An ordinal variable can be discrete or continuous
- Order is important, e.g., rank (e.g., freshman, sophomore, junior, senior)
- Can be treated like interval-scaled
  - Replace *an ordinal variable value* by its rank: $r_{if} \in \{1,\ldots,M_f\}$
  - Map the range of each variable onto [0, 1] by replacing *i*-th object in the *f*-th variable by $z_{if} = \frac{r_{if}-1}{M_f-1}$
  - Example: freshman: 0; sophomore: 1/3; junior: 2/3; senior 1
    - Then distance: d(freshman, senior) = 1, d(junior, senior) = 1/3
  - Compute the dissimilarity using methods for interval-scaled variables

## Cosine Similarity of Two Vectors

- A **document** can be represented by a bag of terms or a long vector, with each attribute recording the *frequency* of a particular term (such as word, keyword, or phrase) in the document

| *Document* | *team* | *coach* | *hockey* | *baseball* | *soccer* | *penalty* | *score* | *win* | *loss* | *season* |
|---|---|---|---|---|---|---|---|---|---|---|
| Document1 | 5 | 0 | 3 | 0 | 2 | 0 | 0 | 2 | 0 | 0 |
| Document2 | 3 | 0 | 2 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| Document3 | 0 | 7 | 0 | 2 | 1 | 0 | 0 | 3 | 0 | 0 |
| Document4 | 0 | 1 | 0 | 0 | 1 | 2 | 2 | 0 | 3 | 0 |

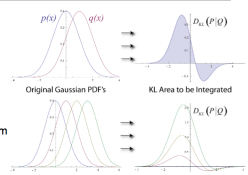
- Other vector objects: Gene features in micro-arrays
- Applications: Information retrieval, biologic taxonomy, gene feature mapping, etc.
- Cosine measure: If $d_1$ and $d_2$ are two vectors (e.g., term-frequency vectors), then

$$cos(d_1,d_2) = \frac{d_1 \bullet d_2}{\|d_1\| \times \|d_2\|}$$

where • indicates vector dot product, $||d||$: the length of vector $d$

## KL Divergence: Comparing Two Probability Distributions

- *The Kullback-Leibler (KL) divergence:* Measure the difference between two probability distributions over the same variable *x*
  - From information theory, closely related to *relative entropy*, *information divergence*, and *information for discrimination*
- $D_{KL}(p(x)\,||\,q(x))$: divergence of $q(x)$ from $p(x)$, measuring the information lost when $q(x)$ is used to approximate $p(x)$




Ack.: Wikipedia entry: *The Kullback-Leibler (KL) divergence*

Discrete form

$$D_{KL}(p(x)||q(x)) = \sum_{x\in X} p(x)\ln\frac{p(x)}{q(x)}$$

Continuous form

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x)\ln\frac{p(x)}{q(x)}dx$$



# Chapter 3. Data Preprocessing

- Redundant attributes may be able to be detected by *correlation analysis* and *covariance analysis*

## Correlation Analysis (for Categorical Data)

- $X^2$ (chi-square) test:

$$\chi^2 = \sum_{i}^{n} \frac{(O_i - E_i)^2}{E_i}$$

(observed: $O_i$; expected: $E_i$)

- Null hypothesis: The two distributions are independent
- The cells that contribute the most to the $X^2$ value are those whose actual count is very different from the expected count
  - The larger the $X^2$ value, the more likely the variables are related
- Note: Correlation does not imply causality
  - # of hospitals and # of car-theft in a city are correlated
  - Both are caused by a third variable: population

## Variance for Single Variable (Numerical Data)

- The variance of a random variable *X* provides a measure of how much the value of *X* deviates from the mean or expected value of *X*:

$$\sigma^2 = \text{var}(X) = E[(X-\mu)^2] = \begin{cases} \sum_x (x-\mu)^2 f(x) & \text{if } X \text{ is discrete} \\ \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

$$\sigma^2 = \text{var}(X) = E[(X-\mu)^2] = E[X^2] - \mu^2 = E[X^2] - [E(x)]^2$$

## Covariance for Two Variables

- Covariance between two variables $X_1$ and $X_2$

$$\sigma_{12} = E[(X_1-\mu_1)(X_2-\mu_2)] = E[X_1X_2] - \mu_1\mu_2 = E[X_1X_2] - E[X_1]E[X_2]$$

where $\mu_1 = E[X_1]$ is the respective mean or **expected value** of $X_1$; similarly for $\mu_2$

- **Positive covariance:** If $\sigma_{12} > 0$
- **Negative covariance**: If $\sigma_{12} < 0$
- **Independence**: If $X_1$ and $X_2$ are independent, $\sigma_{12} = 0$ but the reverse is not true
  - Some pairs of random variables may have a covariance 0 but are not independent
  - Only under some additional assumptions (e.g., the data follow multivariate normal distributions) does a covariance of 0 imply independence

## Correlation between Two Numerical Variables

- **Correlation** between two variables $X_1$ and $X_2$ is the standard covariance, obtained by normalizing the covariance with the standard deviation of each variable

$$\rho_{12} = \frac{\sigma_{12}}{\sigma_1\sigma_2} = \frac{\sigma_{12}}{\sqrt{\sigma_1^2\sigma_2^2}}$$

- **Sample correlation** for two attributes $X_1$ and $X_2$: $\hat{\rho}_{12} = \frac{\hat{\sigma}_{12}}{\hat{\sigma}_1\hat{\sigma}_2} = \frac{\sum_{i=1}^{n}(x_{i1}-\hat{\mu}_1)(x_{i2}-\hat{\mu}_2)}{\sqrt{\sum_{i=1}^{n}(x_{i1}-\hat{\mu}_1)^2\sum_{i=1}^{n}(x_{i2}-\hat{\mu}_2)^2}}$

where n is the number of tuples, $\mu_1$ and $\mu_2$ are the respective means of $X_1$ and $X_2$, $\sigma_1$ and $\sigma_2$ are the respective standard deviation of $X_1$ and $X_2$

- If $\rho_{12} > 0$: A and B are positively correlated ($X_1$'s values increase as $X_2$'s)
  - The higher, the stronger correlation
- If $\rho_{12} = 0$: independent (under the same assumption as discussed in co-variance)
- If $\rho_{12} < 0$: negatively correlated

## Covariance Matrix

- The variance and covariance information for the two variables $X_1$ and $X_2$ can be summarized as 2 X 2 covariance matrix as

$$\Sigma = E[(\mathbf{X}-\mu)(\mathbf{X}-\mu)^T] = E\begin{bmatrix}(X_1-\mu_1)(X_1-\mu_1) & (X_1-\mu_1)(X_2-\mu_2)\\ (X_2-\mu_2)(X_1-\mu_1) & (X_2-\mu_2)(X_2-\mu_2)\end{bmatrix}$$
$$= \begin{pmatrix} E[(X_1-\mu_1)(X_1-\mu_1)] & E[(X_1-\mu_1)(X_2-\mu_2)] \\ E[(X_2-\mu_2)(X_1-\mu_1)] & E[(X_2-\mu_2)(X_2-\mu_2)] \end{pmatrix}$$
$$= \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}$$

- Generalizing it to *d* dimensions, we have,

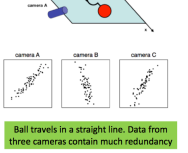
$$D = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & \cdots & x_{dd} \end{pmatrix} \quad \Sigma = E[(\mathbf{X}-\mu)(\mathbf{X}-\mu)^T] = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{pmatrix}$$

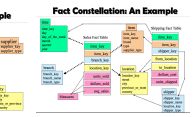
## Data Reduction

regression, histogram,sampling,clustering

## Principal Component Analysis (PCA)

- PCA: A statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called ***principal components***
- The original data are projected onto a much smaller space, resulting in dimensionality reduction
- Method: Find the eigenvectors of the covariance matrix, and these eigenvectors define the new space




Ball travels in a straight line. Data from three cameras contain much redundancy










## Data Cube Measures: Three Categories

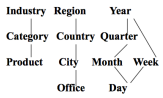
- Distributive: if the result derived by applying the function to *n* aggregate values is the same as that derived by applying the function on all the data without partitioning
  - E.g., count(), sum(), min(), max()
- Algebraic: if it can be computed by an algebraic function with *M* arguments (where *M* is a bounded integer), each of which is obtained by applying a distributive aggregate function
  - avg(x) = sum(x) / count(x)
  - Is min_N() an algebraic measure? How about standard_deviation()?
- Holistic: if there is no constant bound on the storage size needed to describe a subaggregate.
  - E.g., median(), mode(), rank()

## Typical OLAP Operations

- Roll up (drill-up): summarize data
  - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice: *project and select*
- Pivot (rotate):
  - *reorient the cube, visualization, 3D to series of 2D planes*
- Other operations
  - *Drill across: involving (across) more than one fact table*
  - *Drill through: through the bottom level of the cube to its back-end relational tables (using SQL)*

## Efficient Data Cube Computation

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube with L levels? Why this formula? $T = \prod_{i=1}^{n}(L_i+1)$
- Materialization of data cube
  - **Full materialization**: Materialize every (cuboid)
  - **No materialization**: Materialize none (cuboid)
  - **Partial materialization:** Materialize some cuboids
    - Which cuboids to materialize?
      - Selection based on size, sharing, access frequency, etc.




## Indexing OLAP Data: Bitmap Index

- Index on a particular column
  - Each value in the column has a bit vector: bit-op is fast
  - The length of the bit vector: # of records in the base table
  - The *i*-th bit is set if the *i*-th row of the base table has the value for the indexed column
  - not suitable for high cardinality domains
- A recent bit compression technique, Word-Aligned Hybrid (WAH), makes it work for high cardinality domain as well [Wu, et al. TODS'06]

**Base table**

| Cust | Region | Type |
|---|---|---|
| C1 | Asia | Retail |
| C2 | Europe | Dealer |
| C3 | Asia | Dealer |
| C4 | America | Retail |
| C5 | Europe | Dealer |

**Index on Region**

| RecID | Asia | Europe | America |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 |

**Index on Type**

| RecID | Retail | Dealer |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 0 | 1 |
| 4 | 1 | 0 |
| 5 | 0 | 1 |

# Chapter 5. Data Cube Technology

## Cube Materialization: Full Cube vs. Iceberg Cube

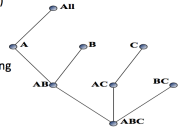
- Full cube vs. iceberg cube
  - compute cube sales iceberg as
  - select month, city, customer group, count(*)
  - from salesInfo
  - cube by month, city, customer group
  - having count(*) >= min support ← iceberg condition
- Compute *only* the cells whose measure satisfies the iceberg condition
- Only a small portion of cells may be "above the water" in a sparse cube
- Ex.: Show only those cells whose count is no less than 100

## Is Iceberg Cube Good Enough? Closed Cube & Cube Shell

- Let cube P have only 2 base cells: $\{(a_1, a_2, a_3 \ldots, a_{100}):10, (a_1, a_2, b_3, \ldots, b_{100}):10\}$
  - How many cells will the iceberg cube contain if "having count(*) ≥ 10"?
    - Answer: $2^{101} - 4$ (still too big!)
- **Close cube:**
  - A cell *c* is ***closed*** if there exists no cell *d*, such that *d* is a descendant of *c*, and *d* has the same measure value as *c*
  - Ex. The same cube P has only 3 closed cells:
    - $\{(a_1, a_2, *, \ldots, *): 20, (a_1, a_2, a_3 \ldots, a_{100}): 10, (a_1, a_2, b_3, \ldots, b_{100}): 10\}$
  - A ***closed cube*** is a cube consisting of only closed cells
- **Cube Shell:** The cuboids involving only a small # of dimensions, e.g., 2
  - Idea: Only compute cube shells, other dimension combinations can be computed on the fly → Q: For $(A_1, A_2, \ldots A_{100})$, how many combinations to compute?

## Multi-Way Array Aggregation

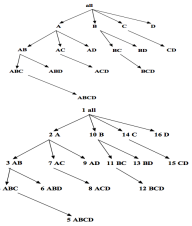
- Array-based "bottom-up" algorithm (from ABC to AB,...)
- Using multi-dimensional chunks
- Simultaneous aggregation on multiple dimensions
- Intermediate aggregate values are re-used for computing ancestor cuboids
- Cannot do *Apriori* pruning: No iceberg optimization




- Comments on the method
  - Efficient for computing the full cube for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods (e.g., BUC) should be used

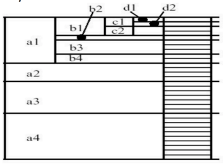
# Cube Computation: Computing in Reverse Order

- BUC (Beyer & Ramakrishnan, SIGMOD'99)
  - BUC: acronym of Bottom-Up (cube) Computation
  - (Note: It is "top-down" in our view since we put Apex cuboid on the top!)
- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *minsup* = 1 ⊳ compute full CUBE!
- No simultaneous aggregation




# BUC: Partitioning and Aggregating

- Usually, entire data set cannot fit in main memory
- Sort *distinct* values
  - partition into blocks that fit
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
  - Collapsing duplicates
    - Cannot do holistic aggregates anymore!




# Shell Fragment Cubes: Size and Design

- Given a database of T tuples, D dimensions, and F shell fragment size, the fragment cubes' space requirement is: $O\left(T\left\lceil\frac{D}{F}\right\rceil(2^F-1)\right)$
  - For F < 5, the growth is sub-linear
- Shell fragments do not have to be disjoint
- Fragment groupings can be arbitrary to allow for maximum online performance
  - Known common combinations (e.g.,<city, state>) should be grouped together
- Shell fragment sizes can be adjusted for optimal balance between offline and online computation

| Attribute Value | TID List | List Size |
|---|---|---|
| a1 | 1 2 3 | 3 |
| a2 | 4 5 | 2 |
| b1 | 1 4 5 | 3 |
| b2 | 2 3 | 2 |
| c1 | 1 2 3 4 5 | 5 |
| d1 | 1 3 4 5 | 4 |
| d2 | 2 | 1 |
| e1 | 1 2 | 2 |
| e2 | 3 4 | 2 |
| e3 | 5 | 1 |

| Cell | Intersection | TID List | List Size |
|---|---|---|---|
| a1 b1 | 1 2 3 ∩ 1 4 5 | 1 | 1 |
| a1 b2 | 1 2 3 ∩ 2 3 | 2 3 | 2 |
| a2 b1 | 4 5 ∩ 1 4 5 | 4 5 | 2 |
| a2 b2 | 4 5 ∩ 2 3 | ϕ | 0 |

$(a_1, a_2, a_3, a_4, \ldots, a_{10}) : 1, (a_1, b_2, a_3, b_4, \ldots, b_{10}) : 1$

i. [3] How many nonempty cuboids are there in this data cube?

**Answer: $2^{10}$.** Since we have 10 dimensions with no concept hierarchy, there are $2^{10}$ cuboids and all of them should not be empty.

ii. [3] How many (nonempty) aggregate closed cells are there in this data cube?

**Answer: 1.** There are 3 closed cells, including the two base cells and $(a_1, *, a_3, *, a_5, *, a_7, *, a_9, *)$. But only the latter one is a **aggregated** closed cell.

iii. [3] How many (nonempty) aggregate cells are there in this data cube?

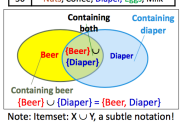
**Answer: 2014.** For each base cell, there are $2^{10}-1$ aggregated cells. However, there are $2^5$ cells that are counted twice since there are 5 common dimensions. Therefore, the total number of nonempty aggregate cells is $2 \cdot (2^{10}-1) - 2^5 = 2014$.

iv. [3] If we set minimum support = 2, how many (nonempty) aggregate cells are there in the corresponding iceberg cube?

**Answer: $2^5$.** These two base cells have common value in 5 dimensions; therefore, there are $2^5$ nonempty cells with support = 2 and all of them are aggregate cells.

# From Frequent Itemsets to Association Rules

| Tid | Items bought |
|---|---|
| 10 | Beer, Nuts, Diaper |
| 20 | Beer, Coffee, Diaper |
| 30 | Beer, Diaper, Eggs |
| 40 | Nuts, Eggs, Milk |
| 50 | Nuts, Coffee, Diaper, Eggs, Milk |




Note: Itemset: X ∪ Y, a subtle notation!

- Association rules: $X \rightarrow Y$ (s, c)
  - Support, *s*: The probability that a transaction contains X ∪ Y
  - Confidence, *c*: The conditional probability that a transaction containing X also contains *Y*
  - c = sup(X ∪ Y) / sup(X)
- **Association rule mining**: Find all of the rules, $X \rightarrow Y$, with minimum support and confidence
- Frequent itemsets: Let *minsup = 50%*
  - Freq. 1-itemsets: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3
  - Freq. 2-itemsets: {Beer, Diaper}: 3
- Association rules: Let *minconf = 50%*
  - *Beer* → *Diaper* (60%, 100%) (Q: Are these all rules?)
  - *Diaper* → *Beer* (60%, 75%)

# Expressing Patterns in Compressed Form: Closed Patterns

- How to handle such a challenge?
- Solution 1: **Closed patterns**: A pattern (itemset) X is closed if X is *frequent*, and there exists *no super-pattern* Y ⊃ X, *with the same support* as X
  - Let Transaction DB $TDB_1$: $T_1$: {$a_1$, ..., $a_{50}$}; $T_2$: {$a_1$, ..., $a_{100}$}
  - Suppose *minsup* = 1. How many closed patterns does $TDB_1$ contain?
    - Two: $P_1$: "{$a_1$, ..., $a_{50}$}: 2"; $P_2$: "{$a_1$, ..., $a_{100}$}: 1"
- Closed pattern is a lossless compression of frequent patterns
  - Reduces the # of patterns but does not lose the support information!
  - You will still be able to say: "{$a_2$, ..., $a_{40}$}: 2", "{$a_5$, $a_{51}$}: 1"

# Expressing Patterns in Compressed Form: Max-Patterns

- Solution 2: **Max-patterns**: A pattern X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ⊃ X
- Difference from close-patterns?
  - Do not care the real support of the sub-patterns of a max-pattern
  - Let Transaction DB $TDB_1$: $T_1$: {$a_1$, ..., $a_{50}$}; $T_2$: {$a_1$, ..., $a_{100}$}
  - Suppose *minsup* = 1. How many max-patterns does $TDB_1$ contain?
    - One: P: "{$a_1$, ..., $a_{100}$}: 1"
- Max-pattern is a lossy compression!
  - We only know {$a_1$, ..., $a_{40}$} is frequent
  - But we do not know the real support of {$a_1$, ..., $a_{40}$}, ..., any more!
- Thus in many applications, mining close-patterns is more desirable than mining max-patterns

# The Apriori Algorithm—An Example

Database TDB (minsup = 2)

| Tid | Items |
|---|---|
| 10 | A, C, D |
| 20 | B, C, E |
| 30 | A, B, C, E |
| 40 | B, E |

$C_1$ (1st scan)

| Itemset | sup |
|---|---|
| {A} | 2 |
| {B} | 3 |
| {C} | 3 |
| {D} | 1 |
| {E} | 3 |

$F_1$

| Itemset | sup |
|---|---|
| {A} | 2 |
| {B} | 3 |
| {C} | 3 |
| {E} | 3 |

$C_2$

| Itemset | sup |
|---|---|
| {A, B} | 1 |
| {A, C} | 2 |
| {A, E} | 1 |
| {B, C} | 2 |
| {B, E} | 3 |
| {C, E} | 2 |

$F_2$

| Itemset | sup |
|---|---|
| {A, C} | 2 |
| {B, C} | 2 |
| {B, E} | 3 |
| {C, E} | 2 |

$C_2$

| Itemset |
|---|
| {A, B} |
| {A, C} |
| {A, E} |
| {B, C} |
| {B, E} |
| {C, E} |

$C_3$

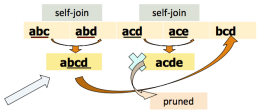
| Itemset |
|---|
| {B, C, E} |

$F_3$

| Itemset | sup |
|---|---|
| {B, C, E} | 2 |

# Apriori: Implementation Tricks

- How to generate candidates?
  - Step 1: self-joining $F_k$
  - Step 2: pruning
- Example of candidate-generation
  - $F_3$ = {*abc, abd, acd, ace, bcd*}
  - Self-joining: $F_3$*$F_3$
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in $F_3$
  - $C_4$ = {*abcd*}




# Exploring Vertical Data Format: ECLAT

- ECLAT (Equivalence Class Transformation): A depth-first search algorithm using set intersection [Zaki et al. @KDD'97]
- Tid-List: List of transaction-ids containing an itemset
- Vertical format: t(e) = {$T_{10}$, $T_{20}$, $T_{30}$}; t(a) = {$T_{10}$, $T_{20}$}; t(ae) = {$T_{10}$, $T_{20}$}
- Properties of Tid-Lists
  - t(X) = t(Y): X and Y always happen together (e.g., t(ac) = t(d))
  - t(X) ⊂ t(Y): transaction having X always has Y (e.g., t(ac) ⊂ t(ce))
- Deriving frequent patterns based on vertical intersections
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - t(e) = {$T_{10}$, $T_{20}$, $T_{30}$}, t(ce) = {$T_{10}$, $T_{30}$} → Diffset (ce, e) = {$T_{20}$}

**A transaction DB in Horizontal Data Format**

| Tid | Itemset |
|---|---|
| 10 | a, c, d, e |
| 20 | a, b, e |
| 30 | b, c, e |

**The transaction DB in Vertical Data Format**

| Item | TidList |
|---|---|
| a | 10, 20 |
| b | 20, 30 |
| c | 10, 30 |
| d | 10 |
| e | 10, 20, 30 |

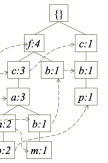
# Example: Construct FP-tree from a Transational DB

| TID | Items in the Transaction | Ordered, frequent items |
|---|---|---|
| 100 | {*f, a, c, d, g, i, m, p*} | {*f, c, a, m, p*} |
| 200 | {*a, b, c, f, l, m, o*} | {*f, c, a, b, m*} |
| 300 | {*b, f, h, j, o, w*} | {*f, b*} |
| 400 | {*b, c, k, s, p*} | {*c, b, p*} |
| 500 | {*a, f, c, e, l, p, m, n*} | {*f, c, a, m, p*} |

1. Scan DB once, find single item frequent pattern: Let min_support = 3
   f:4, a:3, c:4, b:3, m:3, p:3
2. Sort frequent items in frequency descending order, f-list
   F-list = f-c-a-b-m-p
3. Scan DB again, construct FP-tree

**Header Table**

| Item | Frequency | header |
|---|---|---|
| f | 4 | |
| c | 4 | |
| a | 3 | |
| b | 3 | |
| m | 3 | |
| p | 3 | |




# Interestingness Measure: Lift

- Measure of dependent/correlated events: **lift**

$$lift(B,C) = \frac{c(B \rightarrow C)}{s(C)} = \frac{s(B \cup C)}{s(B) \times s(C)}$$

*Lift* is more telling than s & c

| | B | ¬B | $\Sigma_{row}$ |
|---|---|---|---|
| C | 400 | 350 | 750 |
| ¬C | 200 | 50 | 250 |
| $\Sigma_{col}$ | 600 | 400 | 1000 |

- Lift(B, C) may tell how B and C are correlated
  - Lift(B, C) = 1: B and C are independent
  - > 1: positively correlated
  - < 1: negatively correlated
- For our example,

$$lift(B,C) = \frac{400/1000}{600/1000 \times 750/1000} = 0.89$$

$$lift(B,\neg C) = \frac{200/1000}{600/1000 \times 250/1000} = 1.33$$

- Thus, B and C are negatively correlated since lift(B, C) < 1;
  - B and ¬C are positively correlated since lift(B, ¬C) > 1

# Interestingness Measure: $\chi^2$

- Another measure to test correlated events: $\chi^2$

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

| | B | ¬B | $\Sigma_{row}$ |
|---|---|---|---|
| C | 400 (450) | 350 (300) | 750 |
| ¬C | 200 (150) | 50 (100) | 250 |
| $\Sigma_{col}$ | 600 | 400 | 1000 |

Expected value

Observed value

- General rules
  - $\chi^2$ = 0: independent
  - $\chi^2$ > 0: correlated, either positive or negative, so it needs additional test
- Now, $c^2 = \frac{(400-450)^2}{450} + \frac{(350-300)^2}{300} + \frac{(200-150)^2}{150} + \frac{(50-100)^2}{100} = 55.56$
- $\chi^2$ shows B and C are negatively correlated since the expected value is 450 but the observed is only 400
- $\chi^2$ is also more telling than the support-confidence framework

# Interestingness Measures & Null-Invariance

- *Null invariance:* Value does not change with the # of null-transactions
- A few interestingness measures: Some are null invariant

| Measure | Definition | Range | Null-Invariant |
|---|---|---|---|
| $\chi^2(A,B)$ | $\sum_{i,j=0,1} \frac{(e(a_i b_j) - o(a_i b_j))^2}{e(a_i b_j)}$ | $[0, \infty]$ | No |
| $Lift(A,B)$ | $\frac{s(A \cup B)}{s(A) \times s(B)}$ | $[0, \infty]$ | No |
| $AllConf(A,B)$ | $\frac{s(A \cup B)}{max\{s(A), s(B)\}}$ | $[0, 1]$ | Yes |
| $Jaccard(A,B)$ | $\frac{s(A \cup B)}{s(A) + s(B) - s(A \cup B)}$ | $[0, 1]$ | Yes |
| $Cosine(A,B)$ | $\frac{s(A \cup B)}{\sqrt{s(A) \times s(B)}}$ | $[0, 1]$ | Yes |
| $Kulczynski(A,B)$ | $\frac{1}{2}(\frac{s(A \cup B)}{s(A)} + \frac{s(A \cup B)}{s(B)})$ | $[0, 1]$ | Yes |
| $MaxConf(A,B)$ | $max\{\frac{s(A)}{s(A \cup B)}, \frac{s(B)}{s(A \cup B)}\}$ | $[0, 1]$ | Yes |

$\chi^2$ and lift are not null-invariant

Jaccard, consine, AllConf, MaxConf, and Kulczynski are null-invariant measures

# Imbalance Ratio with Kulczynski Measure

- IR (Imbalance Ratio): measure the imbalance of two itemsets A and B in rule implications:

$$IR(A,B) = \frac{|s(A) - s(B)|}{s(A) + s(B) - s(A \cup B)}$$

- Kulczynski and Imbalance Ratio (IR) together present a clear picture for all the three datasets $D_4$ through $D_6$
  - $D_4$ is neutral & balanced; $D_5$ is neutral but imbalanced
  - $D_6$ is neutral but very imbalanced

| Data set | $mc$ | $\neg mc$ | $m \neg c$ | $\neg m \neg c$ | Jaccard | Cosine | Kulc | IR |
|---|---|---|---|---|---|---|---|---|
| $D_1$ | 10,000 | 1,000 | 1,000 | 100,000 | 0.83 | 0.91 | 0.91 | 0 |
| $D_2$ | 10,000 | 1,000 | 1,000 | 100 | 0.83 | 0.91 | 0.91 | 0 |
| $D_3$ | 100 | 1,000 | 1,000 | 100,000 | 0.05 | 0.09 | 0.09 | 0 |
| $D_4$ | 1,000 | 1,000 | 1,000 | 100,000 | 0.33 | 0.5 | 0.5 | 0 |
| $D_5$ | 1,000 | 100 | 10,000 | 100,000 | 0.09 | 0.29 | 0.5 | 0.89 |
| $D_6$ | 1,000 | 10 | 100,000 | 100,000 | 0.01 | 0.10 | 0.5 | 0.99 |

[8] A database has 5 transactions. Let $min\_sup = 0.6$ and $min\_conf = 0.7$.

| trans_id | items_bought |
|---|---|
| 100 | {K, A, D, B, C} |
| 200 | {D, A, E, F} |
| 300 | {C, D, B, E } |
| 400 | {B, A, C, K, D} |
| 500 | {B, G, C} |

i. [2] List the frequent $k$-itemset for the largest $k$, and

**Answer:** $k = 3$
Frequent 3−itemset: $\{B, C, D\}$ with $support = 3/5 = 0.6 \geq min\_sup$.

ii. [6] all the strong association rules (with support and confidence) for the following shape of rules:
$\forall x \in transaction,\ buys(x, item_1) \wedge buys(x, item_2) \Rightarrow buys(x, item_3).$ $[s, c]$

**Answer:**
$\forall x \in transaction,\ buys(x, B) \wedge buys(x, C) \Rightarrow buys(x, D).$ $[0.6, 0.75]$
$\forall x \in transaction,\ buys(x, B) \wedge buys(x, D) \Rightarrow buys(x, C).$ $[0.6, 1]$
$\forall x \in transaction,\ buys(x, C) \wedge buys(x, D) \Rightarrow buys(x, B).$ $[0.6, 1]$

[19] Data Warehousing and OLAP for Data Mining

(a) [9] Suppose the base cuboid of a data cube contains only two cells
$(a_1, a_2, a_3, \ldots, a_{20}), (b_1, b_2, b_3, \ldots, b_{20})$,
where $a_i = b_i$ if $i$ is an odd number; otherwise $a_i \neq b_i$.

i. How many nonempty cuboids are there in this data cube?
**Answer:** $2^{20} = 1,048,576$

ii. How many nonempty aggregate cells are there in this data cube?
**Answer:** Each cell generates $2^{20} - 1$ nonempty aggregate cells. Thus in total we should have $2 \times 2^{20} - 2$ with overlapped cells not considering. We have $2^{10}$ overlapped cells (cells with $a_i$, or $b_i$, fixed where $i$ is odd number). Therefore, we have $2^{21} - 2 - 2^{10} = 2,096,126$.

iii. If we set minimum support = 2, how many nonempty aggregate cells are there in the corresponding iceberg cube?
**Answer:** Only the overlapped cells in the above question have the support 2, and there are no cells that have the support bigger than 2. Therefore, we have $2^{10} = 1,024$.

3. Data Warehousing and OLAP for Data Mining [15'].
Consider a base cuboid of 10 dimensions that contains 4 base cells:

$(a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10})$
$(b_1, a_2, b_3, a_4, b_5, a_6, b_7, a_8, a_9, a_{10})$
$(c_1, a_2, c_3, a_4, c_5, a_6, c_7, a_8, a_9, a_{10})$
$(d_1, a_2, d_3, a_4, d_5, a_6, d_7, a_8, a_9, a_{10})$

where $a_i$, $b_i$, $c_i$ and $d_i$ are distinct for $i = 1, 3, 5, 7$. There is *no* dimension with concept hierarchy. The measure of the cube is *count*. The count of each base cell is 1.

(b) [6'] How many *nonempty aggregated* (i.e, *count* ≥ 1 and non-base) cells does the complete cube contain? Briefly explain your answer.
[ANSWER: First, we consider those cells with at least one of the 1st,3rd,5th,7th dimensions not aggregated(i.e. not *), for example cell $(d_1, *, \ldots, *)$. For each base cell, the number of such cells is $(2^4 - 1) \times 2^6 = 960$. Since we have 4 such base cells, thus the total number of cells in this case is $4 \times 960 = 3840$. Second, we consider those cells with the 1st,3rd,5th,7th dimensions aggregated, for example cell $(*, a_2, *, a_4, *, a_6, *, a_8, ..)$, the number of such cells is $2^6 = 64$. Since all 4 base cells are same if we only consider the 2nd,4th,6th,8th,9th,10th dimensions, the total number of cells in this case is 64. Finally, the total number of non-based cells is $3840 + 64 - 4 = 3900$.]

(c) [5'] How many *nonempty aggregated* cells does an iceberg cube contain, if the condition of the iceberg cube is *count* ≥ 4? Briefly explain.
[ANSWER: Only those cells with the 1st, 3rd, 5th and 7th dimensions aggregated(i.e. *) have count 4. And the number of such cells is $2^6 = 64$]

[16'] Consider a base cuboid of 10 dimensions that contains 4 base cells:

$(a_1, a_2, a_3, a_4, a_5, \ldots, a_{10})$
$(b_1, a_2, a_3, a_4, a_5, \ldots, a_{10})$
$(c_1, c_2, a_3, a_4, a_5, \ldots, a_{10})$
$(d_1, d_2, d_3, a_4, a_5, \ldots, a_{10})$

where $a_i$, $b_i$, $c_i$ and $d_i$ are distinct for $i = 1, 2, 3$. There is no dimension with concept hierarchy. The measure of the cube is count. The count of each base cell is 1.

i. [4'] How many cuboids are there in the full data cube?
ANSWER: $2^{10} = 1024$, **since we have 10 dimensions.**

ii. [6'] How many **nonempty aggregated** cells does the complete cube contain? Briefly explain your answer.
ANSWER: **3196. Firstly, we consider those cells with the 1st dimension not aggregated, for example cell** $(D_1, D_2, D_3, *, \ldots, *)$**. The number of such cells is** $2^9 * 4 = 2048$**. Secondly, we consider those cells with the 1st dimension aggregated but 2nd dimension not aggregated, for example cell** $(*, D_2, D_3, *, \ldots, *)$**. The number of such cells is** $2^8 * 3 = 768$**. Thirdly, we consider those cells with the 1st and 2nd dimension aggregated but 3rd dimension not aggregated, for example cell** $(*, *, D_3, *, \ldots, *)$**. The number of such cells is** $2^7 * 2 = 256$**. Finally, we consider those cells with the 1st, 2nd, and 3rd dimension aggregated, for example cell** $(*, *, *, *, \ldots, *)$**. The number of such cells is** $2^7 * 1 = 128$**. Sum these 4 values and subtract 4 (representing 4 base cells), we get:**$2048 + 768 + 256 + 128 - 4 = 3196$.

ii. [6'] How many **nonempty aggregated** cells does an iceberg cube contain, if the condition of the iceberg cube is *count* ≥ 3? Briefly explain your answer.
ANSWER: **256. Firstly, we note that we must have 1st and 2nd dimension aggregated. Then, if the 3rd dimension is aggregated, we will get** $2^7$ **cells whose support is 4. If the 3rd dimension is not aggregated, we will get** $2^7$ **cells whose support is 3 and** $2^7$ **cells whose support is 1 (these should be neglected). Sum these 2 values, we get:**$2^7 + 2^7 = 2^8 = 256$.

| Constraint | Antimonotonic | Monotonic | Succinct |
|---|---|---|---|
| $v \in S$ | no | yes | yes |
| $S \supseteq V$ | no | yes | yes |
| $S \subseteq V$ | yes | no | yes |
| $min(S) \leq v$ | no | yes | yes |
| $min(S) \geq v$ | yes | no | yes |
| $max(S) \leq v$ | yes | no | yes |
| $max(S) \geq v$ | no | yes | yes |
| $count(S) \leq v$ | yes | no | weakly |
| $count(S) \geq v$ | no | yes | weakly |
| $sum(S) \leq v\ (\forall a \in S, a \geq 0)$ | yes | no | no |
| $sum(S) \geq v\ (\forall a \in S, a \geq 0)$ | no | yes | no |
| $range(S) \leq v$ | yes | no | no |
| $range(S) \geq v$ | no | yes | no |
| $avg(S)\ \theta\ v, \theta \in \{\leq, \geq\}$ | convertible | convertible | no |
| $support(S) \geq \xi$ | yes | no | no |
| $support(S) \leq \xi$ | no | yes | no |
| $all\_confidence(S) \geq \xi$ | yes | no | no |
| $all\_confidence(S) \leq \xi$ | no | yes | no |

Chapter10: Clustering

| | |
|---|---|
| Partitioning Methods | - Find mutually exclusive clusters of **spherical** shape,<br>- Distance based<br>- May use mean or mediod (etc) to represent cluster center<br>- Effective for small - to medium-size data sets |
| Hierarchical Methods | -Clustering is a hierarchical decomposition (i.e. multiple levels)<br>-Cannot correct erroneous merges or splits<br>-May incorporate other techniques like microclustering or consider object "linkages" |
| Density-based methods | -Can find arbitrarily shaped clusters<br>-Clusters are dense regions of objects in space that are separated by low-density regions<br>-Cluster density: Each point must have a minimum number of points within its "neighborhood"<br>-May filter out outliers |
| Grid-based methods | -Use a multiresolution grid data structure<br>-Fast processing time (typically independent of the number of data objects, yet dependent on grid size) |

## Partitioning Algorithms: Basic Concepts

- Partitioning method: Discovering the groupings in the data by optimizing a specific objective function and iteratively improving the quality of partitions
- K-partitioning method: Partitioning a dataset ***D*** of ***n*** objects into a set of ***K*** clusters so that an objective function is optimized (e.g., the sum of squared distances is minimized, where $c_k$ is the centroid or medoid of cluster $C_k$)
  - A typical objective function: **Sum of Squared Errors (SSE)**
    $$SSE(C) = \sum_{k=1}^{K} \sum_{x_i \in C_k} \| x_i - c_k \|^2$$
- Problem definition: Given *K*, find a partition of *K clusters* that optimizes the chosen partitioning criterion
  - Global optimal: Needs to exhaustively enumerate all partitions
  - Heuristic methods (i.e., greedy algorithms): *K-Means*, *K-Medians*, *K-Medoids*, etc.

## The *K-Means* Clustering Method

- *K-Means* (MacQueen'67, Lloyd'57/'82)
  - Each cluster is represented by the center of the cluster
- Given K, the number of clusters, the *K-Means* clustering algorithm is outlined as follows
  - Select *K* points as initial centroids
  - **Repeat**
    - Form *K* clusters by assigning each point to its closest centroid
    - Re-compute the centroids (i.e., *mean point*) of each cluster
  - **Until** convergence criterion is satisfied
- Different kinds of measures can be used
  - Manhattan distance ($L_1$ norm), Euclidean distance ($L_2$ norm), Cosine similarity

## Handling Outliers: From *K-Means* to *K-Medoids*

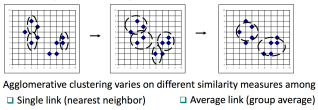
- The *K-Means* algorithm is sensitive to outliers!—since an object with an extremely large value may substantially distort the distribution of the data
- *K-Medoids*: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster
- The *K-Medoids* clustering algorithm:
  - Select *K* points as the initial representative objects (i.e., as initial *K medoids*)
  - **Repeat**
    - Assigning each point to the cluster with the closest medoid
    - Randomly select a non-representative object $o_i$
    - Compute the total cost *S* of swapping the medoid *m* with $o_i$
    - If $S < 0$, then swap *m* with $o_i$ to form the new set of medoids
  - **Until** convergence criterion is satisfied

## K-Modes: Clustering Categorical Data

- *K-Means* cannot handle non-numerical (categorical) data
  - Mapping categorical value to 1/0 cannot generate quality clusters for high-dimensional data
- ***K-Modes***: An extension to *K-Means* by replacing means of clusters with ***modes***
- Dissimilarity measure between object X and the center of a cluster Z
  - $\Phi(x_j, z_j) = 1 - n_j'/n_l$ when $x_j = z_j$ ; 1 when $x_j \neq z_j$
  - where $z_j$ is the categorical value of attribute $j$ in $Z_l$, $n_j$ is the number of objects in cluster $l$, and $n_j'$ is the number of objects whose attribute value is r
- This dissimilarity measure (distance function) is **frequency-based**
- Algorithm is still based on iterative *object cluster assignment* and *centroid update*
- A ***fuzzy K-Modes*** method is proposed to calculate a ***fuzzy cluster membership value*** for each object to each cluster
- A mixture of categorical and numerical data: Using a ***K-Prototype*** method

## Agglomerative Clustering Algorithm

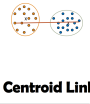
- AGNES (AGglomerative NESting) (Kaufmann and Rousseeuw, 1990)
  - Use the **single-link** method and the dissimilarity matrix
  - Continuously merge nodes that have the least dissimilarity
  - Eventually all nodes belong to the same cluster



- Agglomerative clustering varies on different similarity measures among clusters
  - Single link (nearest neighbor)
  - Average link (group average)
  - Complete link (diameter)
  - Centroid link (centroid similarity)

## Single Link vs. Complete Link in Hierarchical Clustering

- Single link (nearest neighbor)
  - The similarity between two clusters is the similarity between their most similar (nearest neighbor) members
  - Local similarity-based: Emphasizing more on close regions, ignoring the overall structure of the cluster
  - Capable of clustering non-elliptical shaped group of objects
  - Sensitive to noise and outliers
- Complete link (diameter)
  - The similarity between two clusters is the similarity between their most dissimilar members
  - Merge two clusters to form one with the smallest diameter
  - Nonlocal in behavior, obtaining compact shaped clusters
  - Sensitive to outliers

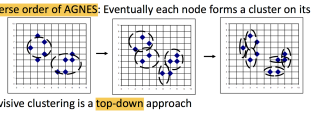


## Agglomerative Clustering: Average vs. Centroid Links

- Agglomerative clustering with **average link**
  - **Average link**: The average distance between an element in one cluster and an element in the other (i.e., all pairs in two clusters)
  - Expensive to compute
- Agglomerative clustering with **centroid link**
  - **Centroid link**: The distance between the centroids of two clusters
- **Group Averaged Agglomerative Clustering (GAAC)**
  - Let two clusters $C_a$ and $C_b$ be merged into $C_{a\cup b}$. The new centroid is: $c_{a\cup b} = \frac{N_a c_a + N_b c_b}{N_a + N_b}$
  - $N_a$ is the cardinality of cluster $C_a$, and $c_a$ is the centroid of $C_a$
  - The similarity measure for GAAC is the average of their distances
- Agglomerative clustering with **Ward's criterion**
  - **Ward's criterion**: The increase in the value of the SSE criterion for the clustering obtained by merging them into $C_a \cup C_b$: $W(C_{a\cup b}, c_{a\cup b}) - W(C, c) = \frac{N_a N_b}{N_a + N_b} d(c_a, c_b)$

## Divisive Clustering

- DIANA (Divisive Analysis) (Kaufmann and Rousseeuw,1990)
  - Implemented in some statistical analysis packages, e.g., Splus
- Inverse order of AGNES: Eventually each node forms a cluster on its own



- Divisive clustering is a top-down approach
  - The process starts at the root with all the points as one cluster
  - It recursively splits the higher level clusters to build the dendrogram
  - Can be considered as a global approach
  - More efficient when compared with agglomerative clustering

## Typical Clustering Methodologies (I)

- **Distance-based methods**
  - Partitioning algorithms: K-Means, K-Medians, K-Medoids
  - Hierarchical algorithms: Agglomerative vs. divisive methods
- **Density-based and grid-based methods**
  - Density-based: Data space is explored at a high-level of granularity and then post-processing to put together dense regions into an arbitrary shape
  - Grid-based: Individual regions of the data space are formed into a grid-like structure
- **Probabilistic and generative models:** Modeling data from a generative process
  - Assume a specific form of the generative model (e.g., mixture of Gaussians)
  - Model parameters are estimated with the Expectation-Maximization (EM) algorithm (using the available dataset, for a maximum likelihood fit)
  - Then estimate the generative probability of the underlying data points

## Clustering Different Types of Data (I)

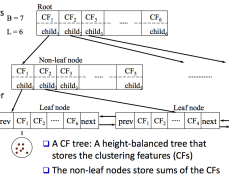
- **Numerical data**
  - Most earliest clustering algorithms were designed for numerical data
- **Categorical data** (including binary data)
  - Discrete data, no natural order (e.g., sex, race, zip-code, and market-basket)
- **Text data**: Popular in social media, Web, and social networks
  - Features: High-dimensional, sparse, value corresponding to word frequencies
  - Methods: Combination of k-means and agglomerative; topic modeling; co-clustering
- **Multimedia data**: Image, audio, video (e.g., on Flickr, YouTube)
  - Multi-modal (often combined with text data)
  - Contextual: Containing both behavioral and contextual attributes
    - Images: Position of a pixel represents its context, value represents its behavior
    - Video and music data: Temporal ordering of records represents its meaning
- **Time-series data**: Sensor data, stock markets, temporal tracking, forecasting, etc.
  - Data are temporally dependent
  - Time: contextual attribute; data value: behavioral attribute
  - Correlation-based online analysis (e.g., online clustering of stock to find stock tickers)
  - Shape-based offline analysis (e.g., cluster ECG based on overall shapes)
- **Sequence data**: Weblogs, biological sequences, system command sequences
  - Contextual attribute: Placement (rather than time)
  - Similarity functions: Hamming distance, edit distance, longest common subsequence
  - Sequence clustering: Suffix tree; generative model (e.g., Hidden Markov Model)
- **Stream data**:
  - Real-time, evolution and concept drift, single pass algorithm
  - Create efficient intermediate representation, e.g., micro-clustering

## BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies)

- A multiphase clustering algorithm (Zhang, Ramakrishnan & Livny, SIGMOD'96)
- Incrementally construct a CF (Clustering Feature) tree, a hierarchical data structure for multiphase clustering
  - Phase 1: Scan DB to build an initial in-memory CF tree (a multi-level compression of the data that tries to preserve the inherent clustering structure of the data)
  - Phase 2: Use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree
- Key idea: Multi-level clustering
  - Low-level micro-clustering: Reduce complexity and increase scalability
  - High-level macro-clustering: Leave enough flexibility for high-level clustering
- *Scales linearly*: Find a good clustering with a single scan and improve the quality with a few additional scans

## The CF Tree Structure in BIRCH

- Incremental insertion of new points (similar to B+-tree)
- For each point in the input
  - Find closest leaf entry
  - Add point to leaf entry and update CF
  - If entry diameter > max_diameter
    - split leaf, and possibly parents
- A CF tree has two parameters
  - Branching factor: Maximum number of children
  - Maximum diameter of sub-clusters stored at the leaf nodes




- A CF tree: A height-balanced tree that stores the clustering features (CFs)
- The non-leaf nodes store sums of the CFs of their children

## BIRCH: A Scalable and Flexible Clustering Method

- An integration of agglomerative clustering with other (flexible) clustering methods
  - Low-level micro-clustering
    - Exploring CP-feature and BIRCH tree structure
    - Preserving the inherent clustering structure of the data
  - Higher-level macro-clustering
    - Provide sufficient flexibility for integration with other clustering methods
- Impact to many other clustering methods and applications
- Concerns
  - Sensitive to insertion order of data points
  - Due to the fixed size of leaf nodes, clusters may not be so natural
  - Clusters tend to be spherical given the radius and diameter measures

## CHAMELEON: Hierarchical Clustering Using Dynamic Modeling

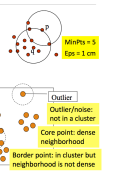
- CHAMELEON: A graph partitioning approach (G. Karypis, E. H. Han, and V. Kumar, 1999)
- Measures the similarity based on a dynamic model
  - Two clusters are merged only if the *interconnectivity* and *closeness (proximity)* between two clusters are high *relative to* the internal interconnectivity of the clusters and closeness of items within the clusters
- A graph-based, two-phase algorithm
  1. Use a graph-partitioning algorithm: Cluster objects into a large number of relatively small sub-clusters
  2. Use an agglomerative hierarchical clustering algorithm: Find the genuine clusters by repeatedly combining these sub-clusters

## Density-Based Clustering Methods

- Clustering based on density (a local cluster criterion), such as density-connected points
- Major features:
  - Discover clusters of arbitrary shape
  - Handle noise
  - One scan (only examine the local region to justify density)
  - Need density parameters as termination condition

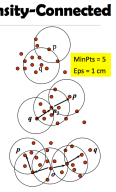
## DBSCAN: A Density-Based Spatial Clustering Algorithm

- DBSCAN (M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, KDD'96)
  - Discovers clusters of arbitrary shape: Density-Based Spatial Clustering of Applications with Noise
- A *density-based* notion of cluster
  - A *cluster* is defined as a maximal set of density-connected points
- Two parameters:
  - *Eps* (ε): Maximum radius of the neighborhood
  - *MinPts*: Minimum number of points in the Eps-neighborhood of a point
  - The Eps(ε)-neighborhood of a point *q*:
    - $N_{Eps}(q)$: {p belongs to D | dist(p, q) ≤ Eps}



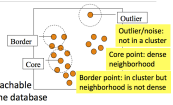

## DBSCAN: Density-Reachable and Density-Connected

- **Directly density-reachable**:
  - A point *p* is directly density-reachable from a point *q* w.r.t. *Eps* (ε), *MinPts* if
    - *p* belongs to $N_{Eps}(q)$
    - **core point** condition: $|N_{Eps}(q)| \geq MinPts$
- **Density-reachable**:
  - A point *p* is density-reachable from a point *q* w.r.t. *Eps*, *MinPts* if there is a chain of points $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ such that $p_{i+1}$ is directly density-reachable from $p_i$
- **Density-connected**:
  - A point *p* is density-connected to a point *q* w.r.t. *Eps*, *MinPts* if there is a point *o* such that both *p* and *q* are density-reachable from *o* w.r.t. *Eps* and *MinPts*



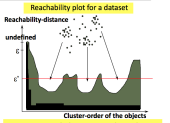

## DBSCAN: The Algorithm

- **Algorithm**
  - Arbitrarily select a point *p*
  - Retrieve all points density-reachable from *p* w.r.t. *Eps* and *MinPts*
  - If *p* is a core point, a cluster is formed
  - If *p* is a border point, no points are density-reachable from *p*, and DBSCAN visits the next point of the database
  - Continue the process until all of the points have been processed
- **Computational complexity**
  - If a spatial index is used, the computational complexity of DBSCAN is O(nlogn), where n is the number of database objects
  - Otherwise, the complexity is $O(n^2)$




## OPTICS: Ordering Points To Identify Clustering Structure

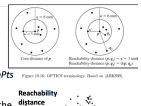
- OPTICS (Ankerst, Breunig, Kriegel, and Sander, SIGMOD'99)
  - DBSCAN is sensitive to parameter setting
  - An extension: finding clustering structure
- Observation: Given a *MinPts*, density-based clusters w.r.t. a higher density are completely contained in clusters w.r.t. to a lower density
- Idea: Higher density points should be processed first—find high-density clusters first
- OPTICS stores such a clustering order using two pieces of information:
  - *Core distance* and *reachability distance*




- Since points belonging to a cluster have a low reachability distance to their nearest neighbor, valleys correspond to clusters
- The deeper the valley, the denser the cluster

## OPTICS: An Extension from DBSCAN

- **Core distance** of an object *p*: The smallest value ε such that the ε-neighborhood of *p* has at least *MinPts* objects
  - Let $N_\varepsilon(p)$: ε-neighborhood of *p*
  - ε is a distance value
  - Core-distance$_{\varepsilon, MinPts}(p)$ = Undefined if card($N_\varepsilon(p)$) < *MinPts*
    *MinPts*-distance(p), otherwise
- **Reachability distance** of object *p* from core object *q* is the min. radius value that makes *p* density-reachable from *q*
  - Reachability-distance$_{\varepsilon, MinPts}(p, q)$ =
    Undefined, *if q is not a core object*
    max(core-distance(*q*), distance (*q*, *p*)), *otherwise*
- Complexity: O(*N* logN) (if index-based)
  where *N*: # of points

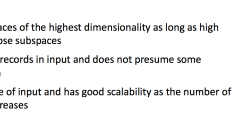



## Grid-Based Clustering Methods

- Grid-based Clustering: Explore multi-resolution grid data structure in clustering
  - Partition the data space into a finite number of cells to form a grid structure
  - Find clusters (dense regions) from the cells in the grid structure
- Features and challenges of a typical grid-based algorithm
  - Efficiency and scalability: # of cells << # of data points
  - Uniformity: Uniform, hard to handle highly irregular data distributions
  - Locality: Limited by predefined cell sizes, borders, and the density threshold
  - Curse of dimensionality: Hard to cluster high-dimensional data
- Methods to be introduced
  - **STING** (a STatistical INformation Grid approach) (Wang, Yang and Muntz, VLDB'97)
  - **CLIQUE** (Agrawal, Gehrke, Gunopulos, and Raghavan, SIGMOD'98)
    - Both grid-based and subspace clustering
  - Advantages
    - Query-independent, easy to parallelize, incremental update
    - Efficiency: Complexity is O(K)
      - K: # of grid cells at the lowest level, and K << N (i.e., # of data points)
  - Disadvantages
    - Its probabilistic nature may imply a loss of accuracy in query processing
- **Grid-based**: It discretizes the data space through a grid and estimates the density by counting the number of points in a grid cell
- **Density-based**: A cluster is a maximal set of connected dense units in a subspace
  - A unit is dense if the fraction of total data points contained in the unit exceeds the input model parameter
- **Subspace clustering**: A subspace cluster is a set of neighboring dense cells in an arbitrary subspace. It also discovers some minimal descriptions of the clusters

## STING: A Statistical Information Grid Approach

- STING (Statistical Information Grid) (Wang, Yang and Muntz, VLDB'97)
- The spatial area is divided into rectangular cells at different levels of resolution, and these cells form a tree structure
- A cell at a high level contains a number of smaller cells of the next lower level
- Statistical information of each cell is calculated and stored beforehand and is used to answer queries
- Parameters of higher level cells can be easily calculated from that of lower level cell, including
  - *count*, *mean*, *s*(standard deviation), *min*, *max*
  - type of distribution—*normal*, *uniform*, etc.




## Additional Comments on *CLIQUE*

- Strengths
  - *Automatically* finds subspaces of the highest dimensionality as long as high density clusters exist in those subspaces
  - *Insensitive* to the order of records in input and does not presume some canonical data distribution
  - Scales *linearly* with the size of input and has good scalability as the number of dimensions in the data increases
- Weaknesses
  - As in all grid-based clustering approaches, the quality of the results crucially depends on the appropriate choice of the number and width of the partitions and grid cells

- **External**: Supervised, employ criteria not inherent to the dataset
  - Compare a clustering against prior or expert-specified knowledge (i.e., the ground truth) using certain clustering quality measure
- **Internal**: Unsupervised, criteria derived from data itself
  - Evaluate the goodness of a clustering by considering how well the clusters are separated and how compact the clusters are, e.g., silhouette coefficient
- **Relative**: Directly compare different clusterings, usually those obtained via different parameter settings for the same algorithm

Chapter7_____

## Different Kinds of Constraints Lead to Different Pruning Strategies

- Constraints can be categorized as
  - Pattern space pruning constraints vs. data space pruning constraints
- Pattern space pruning constraints
  - Anti-monotonic: If constraint *c* is violated, its further mining can be terminated
  - Monotonic: If *c* is satisfied, no need to check *c* again
  - Succinct: If the constraint *c* can be enforced by directly manipulating the data
  - Convertible: *c* can be converted to monotonic or anti-monotonic if items can be properly ordered in processing
- Data space pruning constraints
  - Data succinct: Data space can be pruned at the initial pattern mining process
  - Data anti-monotonic: If a transaction *t* does not satisfy *c*, then *t* can be pruned to reduce data processing effort

## Pattern Space Pruning with Pattern Anti-Monotonicity

- Constraint c is *anti-monotone*
  - If an itemset S **violates** constraint c, so does any of its superset
  - That is, mining on itemset S can be terminated
- Ex. 1: $c_1$: *sum(S.price)* ≤ *v* is anti-monotone
- Ex. 2: $c_2$: range(S.profit) ≤ 15 is anti-monotone
  - Itemset *ab* violates $c_2$ (range(ab) = 40)
  - So does every superset of *ab*
- Ex. 3. $c_3$: *sum(S.Price)* ≥ *v* is not anti-monotone
- Ex. 4. Is $c_4$: *support(S)* ≥ *σ* anti-monotone?
  - Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!

| TID | Transaction |
|---|---|
| 10 | a, b, c, d, f, h |
| 20 | b, c, d, f, g, h |
| 30 | b, c, d, f, g |
| 40 | a, c, e, f, g |

min_sup = 2
price(item)>0

| Item | Profit |
|---|---|
| a | 40 |
| b | 0 |
| c | -20 |
| d | -15 |
| e | -30 |
| f | -10 |
| g | 20 |
| h | 5 |

## Pattern Monotonicity and Its Roles

- A constraint *c* is *monotone*: If an itemset S **satisfies** the constraint *c*, so does any of its superset
  - That is, we do not need to check *c* in subsequent mining
- Ex. 1: $c_1$: *sum(S.Price)* ≥ *v* is monotone
- Ex. 2: $c_2$: *min(S.Price)* ≤ *v* is monotone
- Ex. 3: $c_3$: range(S.profit) ≥ 15 is monotone
  - Itemset *ab* satisfies $c_3$
  - So does every superset of *ab*

| TID | Transaction |
|---|---|
| 10 | a, b, c, d, f, h |
| 20 | b, c, d, f, g, h |
| 30 | b, c, d, f, g |
| 40 | a, c, e, f, g |

min_sup = 2
price(item)>0

| Item | Profit |
|---|---|
| a | 40 |
| b | 0 |
| c | -20 |
| d | -15 |
| e | -30 |
| f | -10 |
| g | 20 |
| h | 5 |

## Data Space Pruning with Data Anti-Monotonicity

- A constraint c is *data anti-monotone*: In the mining process, if a data entry *t* cannot satisfy a pattern *p* under *c*, *t* cannot satisfy *p*'s superset either
  - Data space pruning: Data entry *t* can be pruned
- Ex. 1: $c_1$: *sum(S.Profit)* ≥ *v* is data anti-monotone
  - Let constraint $c_1$ be: sum(S.Profit) ≥ 25
  - $T_{30}$: {b, c, d, f, g} can be removed since none of their combinations can make an S whose sum of the profit is ≥ 25
- Ex. 2: $c_2$: *min(S.Price)* ≤ *v* is data anti-monotone
  - Consider *v* = 5 but every item in transaction $T_{50}$ has a price higher than 10
- Ex. 3: $c_3$: *range(S.Profit)* ≥25 is data anti-monotone

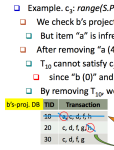
| TID | Transaction |
|---|---|
| 10 | a, b, c, d, f, h |
| 20 | b, c, d, f, g, h |
| 30 | b, c, d, f, g |
| 40 | a, c, e, f, g |

min_sup = 2
price(item)>0

| Item | Profit |
|---|---|
| a | 40 |
| b | 0 |
| c | -20 |
| d | -15 |
| e | -30 |
| f | -10 |
| g | 20 |
| h | 5 |

## Data Space Pruning Should Be Explored Recursively

- Example. $c_4$: *range(S.Profit)* > 25
  - We check b's projected database
  - But item "a" is infrequent (sup = 1)
  - After removing "a (40)" from $T_{10}$
  - $T_{10}$ cannot satisfy $c_4$ any more
    - since "b (0)" and "c (-20), d (-15), f (-10), h (5)"
  - By removing $T_{10}$, we can also prune "h" in $T_{20}$




- Note: $c_4$ prunes $T_{10}$ effectively only after "a" is pruned (by min-sup) in b's projected DB

## Mining Compressed Patterns

| Pat-ID | Item-Sets | Support |
|---|---|---|
| P1 | {38,16,18,12} | 205227 |
| P2 | {38,16,18,12,17} | 205211 |
| P3 | {39,36,16,18,12,17} | 101758 |
| P4 | {39,16,18,12,17} | 161563 |
| P5 | {39,16,18,12} | 161576 |

- Closed patterns
  - P1, P2, P3, P4, P5
  - Emphasizes too much on support
  - There is no compression
- Max-patterns
  - P3: information loss
- Desired output (a good balance):
  - P2, P3, P4

- Why mining compressed patterns?
  - Too many scattered patterns but not so meaningful
- Pattern distance measure
  $$Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}$$
- δ-clustering: For each pattern P, find all patterns which can be expressed by P and whose distance to P is within δ (δ-cover)
- All patterns in the cluster can be represented by P
- Method for efficient, direct mining of compressed frequent patterns (e.g., D. Xin, J. Han, X. Yan, H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60:5-29, 2007)

- Challenges of mining long patterns
  - The curse of "downward closure" property of frequent patterns
  - Any sub-pattern of a frequent pattern is frequent
  - If $\{a_1, a_2, ..., a_{100}\}$ is frequent, then $\{a_1\}$, $\{a_2\}$, ..., $\{a_{100}\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, ..., $\{a_1, a_{100}\}$, $\{a_1, a_2, a_3\}$, ... are all frequent! There are about $2^{100}$ such frequent itemsets!
  - No matter searching in breadth-first (e.g., Apriori) or depth-first (e.g., FPgrowth), if we still adopt the "small to large" step-by-step growing paradigm, we have to examine so many patterns, which leads to combinatorial explosion!

## The Pattern-Fusion Algorithm

- Initialization (Creating initial pool): Use an existing algorithm to mine all frequent patterns up to a small size, e.g., 3
- Iteration (Iterative Pattern Fusion):
  - At each iteration, K seed patterns are randomly picked from the current pattern pool
  - For each seed pattern thus picked, we find all the patterns within a bounding ball centered at the seed pattern
  - All these patterns found are fused together to generate a set of super-patterns
  - All the super-patterns thus generated form a new pool for the next iteration
- Termination: when the current pool contains no more than K patterns at the beginning of an iteration

## GSP: Apriori-Based Sequential Pattern Mining

- Repeat (for each level (i.e., length-k))
  - Scan DB to find length-k frequent sequences
  - Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori
  - set k = k+1
- Until no frequent sequence or no candidate can be found

## Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

## PrefixSpan: A Pattern-Growth Approach

| SID | Sequence |
|---|---|
| 10 | <a[abc](ac)d(cf)> |
| 20 | <(ad)c(bc)(ae)> |
| 30 | <(ef)(ab)(df)cb> |
| 40 | <eg(af)cbc> |

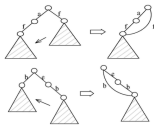
| Prefix | Suffix (Projection) |
|---|---|
| <a> | <(abc)(ac)d(cf)> |
| <aa> | <(_bc)(ac)d(cf)> |
| <ab> | <(_c)(ac)d(cf)> |

- Prefix and suffix
- Given <a(abc)(ac)d(cf)>
- Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, ...
- Suffix: Prefixes-based projection

- PrefixSpan Mining: Prefix Projections
  - Step 1: Find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: Divide search space and mine each projected DB

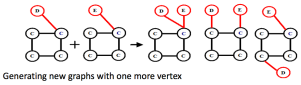
## CloSpan: Mining Closed Sequential Patterns

- A closed sequential pattern *s*: There exists no superpattern *s'* such that *s'* ⊃ *s*, and *s'* and *s* have the same support
- Which ones are closed? <abc>: 20, <abcd>:20, <abcde>: 15
- Why directly mine closed sequential patterns?
  - Reduce # of (redundant) patterns
  - Attain the same expressive power
- Property $P_2$: If *s* ⊃ $s_1$, *s* is closed iff two project DBs have the same size
- Explore *Backward Subpattern* and *Backward Superpattern* pruning to prune redundant search space
- Greatly enhances efficiency (Yan, et al., SDM'03)



## Candidate Generation: Vertex Growing vs. Edge Growing

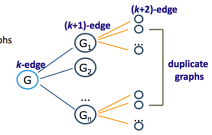
- Methodology: breadth-search, Apriori joining two size-*k* graphs
  - Many possibilities at generating size-(*k*+1) candidate graphs



- Generating new graphs with one more vertex
  - AGM (Inokuchi, et al., PKDD'00)
- Generating new graphs with one more edge
  - FSG (Kuramochi and Karypis, ICDM'01)
- Performance shows *via edge growing* is more efficient

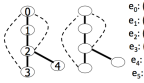
## Pattern-Growth Approach

- Depth-first growth of subgraphs from *k*-edge to (*k*+1)-edge, then (*k*+2)-edge subgraphs
- Major challenge
  - Generating many duplicate subgraphs
- Major idea to solve the problem
  - Define an order to generate subgraphs
  - DFS spanning tree: Flatten a graph into a sequence using depth-first search
  - gSpan (Yan & Han: ICDM'02)




## gSPAN: Graph Pattern Growth in Order

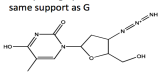
- **Right-most path extension** in subgraph pattern growth
  - Right-most path: The path from root to the right-most leaf (choose the vertex w. the smallest index at each step)
  - Reduce generation of duplicate subgraphs
- **Completeness:** The Enumeration of graphs using right-most path extension is complete
- DFS Code: Flatten a graph into a sequence using depth-first search



$e_0$: (0,1)
$e_1$: (1,2)
$e_2$: (2,0)
$e_3$: (2,3)
$e_4$: (3,0)
$e_5$: (2,4)

## Why Mining Closed Graph Patterns?

- Challenge: An *n*-edge frequent graph may have $2^n$ subgraphs
- Motivation: Explore *closed frequent subgraphs* to handle graph pattern explosion problem
- A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G



If this subgraph is *closed* in the graph dataset, it implies that none of its frequent super-graphs carries the same support

- *Lossless compression*: Does not contain non-closed graphs, but still ensures that the mining result is complete
- Algorithm CloseGraph: Mines closed graph patterns directly

**Chapter 8**

## Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let $p_i$ be the probability that an arbitrary tuple in D belongs to class $C_i$, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

## Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

| age | $p_i$ | $n_i$ | I($p_i$, $n_i$) |
|---|---|---|---|
| <=30 | 2 | 3 | 0.971 |
| 31…40 | 4 | 0 | 0 |
| >40 | 3 | 2 | 0.971 |

$\frac{5}{14} I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's.

Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

14

## Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

- GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex. $SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$
  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

## Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where $p_j$ is the relative frequency of class *j* in *D*

- If a data set *D* is split on A into two subsets $D_1$ and $D_2$, the *gini* index *gini*(*D*) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

## Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain:**
    - biased towards multivalued attributes
  - **Gain ratio:**
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index:**
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

## Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

## Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

| A\P | C | ¬C | |
|---|---|---|---|
| C | TP | FN | P |
| ¬C | FP | TN | N |
| | P' | N' | All |

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified
  - **Accuracy = (TP + TN)/All**
- **Error rate:** *1 – accuracy*, or
  - **Error rate = (FP + FN)/All**
- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

## Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness: what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Comment:

$$recall = \frac{TP}{TP + FN}$$

  - Perfect score is 1.0
  - Inverse relationship between precision & recall
- ***F* measure** (or ***F*-score**): harmonic mean of precision and recall
  - In general, it is the weighted measure of precision & recall

$$F = \frac{1}{\alpha \cdot \frac{1}{P} + (1-\alpha) \cdot \frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

Assigning β times as much weight to recall as to precision)

- ***F1-measure (balanced F-measure)***
  - That is, when β = 1,

$$F_1 = \frac{2PR}{P+R}$$

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use $D_i$ as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation****: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data
- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - Each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

## Adaboost (Freund and Schapire, 1997)

- Given a set of *d* class-labeled tuples, $(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_d, y_d)$
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set $D_i$ of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model $M_i$ is derived from $D_i$
  - Its error rate is calculated using $D_i$ as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: $err(\mathbf{X}_j)$ is the misclassification error of tuple $\mathbf{X}_j$. Classifier $M_i$ error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X}_j)$$

- The weight of classifier $M_i$'s vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)

## Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods in two-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: move the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - **Ensemble techniques**: Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

## Chapter 9 : Advanced Frequent Pattern Mining

- If more than one rule are triggered, need **conflict resolution**
  - **Size ordering**: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute tests*)
  - **Class-based ordering**: decreasing order of *prevalence or misclassification cost per class*
  - **Rule-based ordering (decision list)**: rules are organized into one long priority list, according to some measure of rule quality or by experts

## Rule Extraction from a Decision Tree

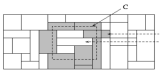
- Rules are *easier to understand* than large trees
- One rule is created *for each path* from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive


**Bayesian Network** =>based on conditional independence
**Neural Network** => skip **Discriminative Classifier** => Pro: prediction accy higher than bayesian; robust when training contains error; fast evaluation of target function => Cons: long training time; difficult to understand learning function(bayesian can be used for pattern discovery); not easy to incorporate domain knowledge(prior, distribution)
**SVM**=>for linear and nonlinear data => **w**x+b=0 (w=weight vector) => w0+w1x1+w2x2=0 => why good? Complexity is # of support vector rather than dimension of data; support vector are critical training samples; good generalization even when high dimensionality.
**CF-Tree: Hierarchical Micro clustering** => SVM not scalable, therefore => Clustering based SVM Algorithm: 1. Construct two CF trees. 2. Train SVM from centroid of root entries. 3. De-cluster the entries near the boundary into next level. (child entries declustered from parent are accumulated into training set with non declustered parent entries) 4. Repeat until nothing is accumulated. **Micro Cluster construction**=> One scan of the data set: Construct two CF-trees (i.e., statistical summary of the data) from positive and negative data sets independently =>Micro-clustering: Hierarchical indexing structure (Provide finer samples closer to the boundary and coarser samples farther from the boundary). =>selective Declustering (high accuracy)
**Rule & Pattern Based classification: If-then rule:** => 基本上就是 decision tree 的每条 path => two assessment: coverage & accuracy: ncovers = # of tuples covered by R1; ncorrect = # of tuples correctly classified by R1; **coverage**(R1) = ncovers /|TrainSet|; **accuracy**(R1) = ncorrect / ncovers => if more than one rule resolve conflict: 1.size ordering 2. class based ordering. 3. rule based ordering. **Sequential Covering**: extract rule from training data =>从 data 学 rule，学一个 rule 扔所有可以被 rule 涵盖的 tuple. Sequential Covering **Vs** Decision tree: learning set of rule simultaneously
**Associative or pattern based classification**
**CBA:** classification based on associations => **Method**: 1. mine high-confidence, high-support rules. 2. Assign labels to rules 3. Classification: apply first matched rules, o.w apply default rule.
**CMAR (improve efficiency and accuracy)**: Classification Based on Multiple Association Rules => **Method:** 1.Rule pruning whenever a rule is inserted to tree 2. Classification based on generated rules (if a set of rule satisfies: use chi square to find strongest group of rules)
**Discriminative Pattern based classification:** mining discriminative frequent pattern as high quality features and apply classifier **Framework: PatClass** => feature construction by frequent itemset mining; feature selection (selective discriminative features) => apply general classifier

**Lazy learner: KNN & Case Based Reasoning**: use database of problem solution to solve new problems
**Genetic Algo:** 和生物进化类似，随机生成 initial population，fittset 不断进化，Based on the notion of survival of the fittest, a new population is formed to consist of the fittest rules (就是他妈的准确率) and their offspring
**不同种类的 machine learning**: **Semi Supervised Classification**: Adv: easy understand Disadv: reinforce error
**Active learning:** achieve high accuracy using as few label as possible **Transfer learning**: Extract knowledge from one or more source tasks and apply the knowledge to a target task

## Rough Set Approach

- Rough sets are used to **approximately or "roughly" define equivalent classes**
- A rough set for a given class C is approximated by two sets: a lower approximation (certain to be in C) and an upper approximation (cannot be described as not belonging to C)
- Finding the minimal subsets (**reducts**) of attributes for feature reduction is NP-hard but a **discernibility matrix** (which stores the differences between attribute values for each pair of data tuples) is used to reduce the computation intensity



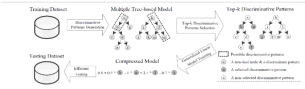

## Case-Based Reasoning (CBR)

- **CBR**: Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
  - Instances represented by rich symbolic descriptions (e.g., function graphs)
  - Search for similar cases, multiple retrieved cases may be combined
  - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
- Challenges
  - Find a good similarity metric
  - Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

## Lazy vs. Eager Learning

- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

## DPClass: Discriminative Pattern-based Classification




- Adopt every prefix path in an (extremely) random forest as a candidate pattern
  - The split points of continuous variables are automatically chosen by random forest → No discretization!
- Run top-k (e.g., top-20) pattern selection based on training data
- Train a generalized linear model (e.g., logistic regression) based on "bag-of-patterns" representations of training data

**Bootstrap:** sampling of the given data set *with replacement*.

**Ensemble Method:** An ensemble is a series of *k* learned models,*M*1,*M*2,...,*Mk* with the aim of creating an improved composite classification model, *M**. The ensemble returns a class predication based on the votes of the base classifiers.

**Bagging:** Bootstrap aggregation. Given a set *D*, of *d* tuples, **bagging** works as follows. For iteration *i*(*i* = 1, 2, ... *k*), a training set, *Di* of *d* tuples is sampled with replacement from the original set of tuples, *D*. Each classifier in an ensemble *M** is learned off of a bootstrapped sample. That is, for *Mi* in *M** it is learned off of *Di*.

**Boosting:** Weights are assigned to each training tuple. A series of *k* classifiers is iteratively learned. After a classifier,*Mi* is learned, the weights are updated to allow the subsequent classifier,*Mi*+1 to "pay more attention" to the training tuples that were misclassified by *Mi*. The final boosted classifier,*M** combines the votes of each individual classifier, where the weight of each classifiers vote is a function of its accuracy.

Give the name of the measure for the distance between two objects for each of the 4 different kinds of data.
Numeric: minkowaski Binary: jaccard
Nomial: cosine(vector)

Give the names of 4 measures for the distance between two clusters for numerical data.
Single Link Complete Link Average Link Centroid Link