Data Mining:

Concepts and Techniques

— Chapter 5 —

Source Slides from Data Mining: Concepts and Techniques

-Jiawei Han and Micheleline Kamber

Chapter 5: Mining Frequent Patterns, Association and Correlations

- Basic concepts and a road map
- Efficient and scalable frequent item set mining methods
- Mining various kinds of association rules

What Is Frequent Pattern Analysis?

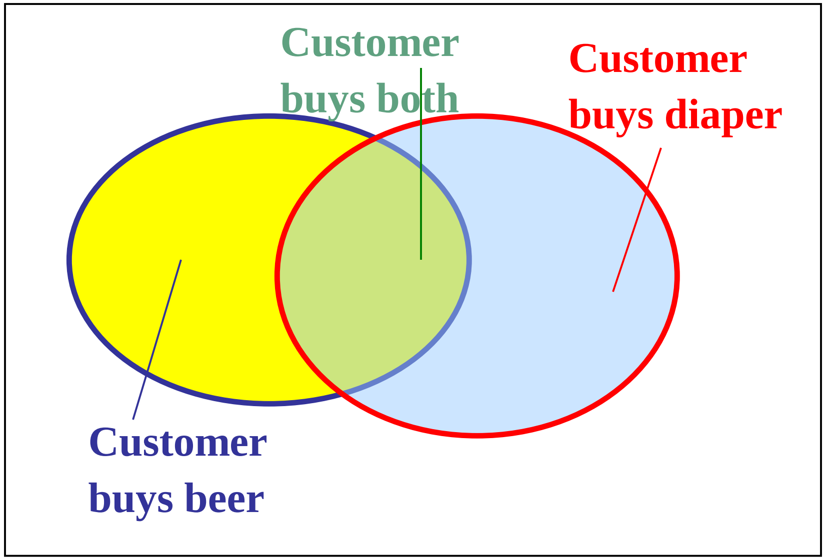
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Freq. Pattern Mining Important?

- Discloses an intrinsic and important property of data sets
- Forms the foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: associative classification
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Concepts: Frequent Patterns and Association Rules

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



- Itemset X = $\{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

Let $sup_{min} = 50\%$, $conf_{min} = 50\%$

Freq. Pat.: {*A:3, B:3, D:4, E:3, AD:3*}

Association rules:

$A \rightarrow D$ (60%, 100%)

$D \rightarrow A$ (60%, 75%)

Closed Patterns and Max-Patterns

- A long pattern contains a combinatorial number of sub-patterns, e.g., $\{a_1, \ldots, a_{100}\}$ contains $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27*10^{30}$ sub-patterns!
- Solution: *Mine closed patterns and max-patterns instead*
- An itemset X is closed if X is *frequent* and there exists *no super-pattern* Y ⊃ X, *with the same support* as X (proposed by Pasquier, et al. @ ICDT'99)
- An itemset X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ⊃ X (proposed by Bayardo @ SIGMOD'98)
- Closed pattern is a lossless compression of freq. patterns
  - Reducing the # of patterns and rules

Closed Patterns and Max-Patterns

- Exercise. DB = $\{<a_1, \ldots, a_{100}>, < a_1, \ldots, a_{50}>\}$
  - Min_sup = 1.
- What is the set of closed itemset?
  - $<a_1, \ldots, a_{100}>$: 1
  - $< a_1, \ldots, a_{50}>$: 2
- What is the set of max-pattern?
  - $<a_1, \ldots, a_{100}>$: 1
- What is the set of all patterns?
  - !!

Chapter 5: Mining Frequent Patterns, Association and Correlations

- Basic concepts and a road map
- Efficient and scalable frequent itemset mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

Scalable Methods for Mining Frequent Patterns

- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

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

1st scan →

C_1

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

L_1

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

C_2

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

2nd scan ←

C_2

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

L_2

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

C_3

Itemset
{B, C, E}

3rd scan →

L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

- Pseudo-code:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items};
  for (k = 1; L_k !=$\emptyset$; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
  increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into $\boldsymbol{C_k}$

  select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

  from $\boldsymbol{L_{k-1}\ p, L_{k-1}\ q}$

  where $\boldsymbol{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$

- Step 2: pruning

  forall ***itemsets c in*** $\boldsymbol{C_k}$ do

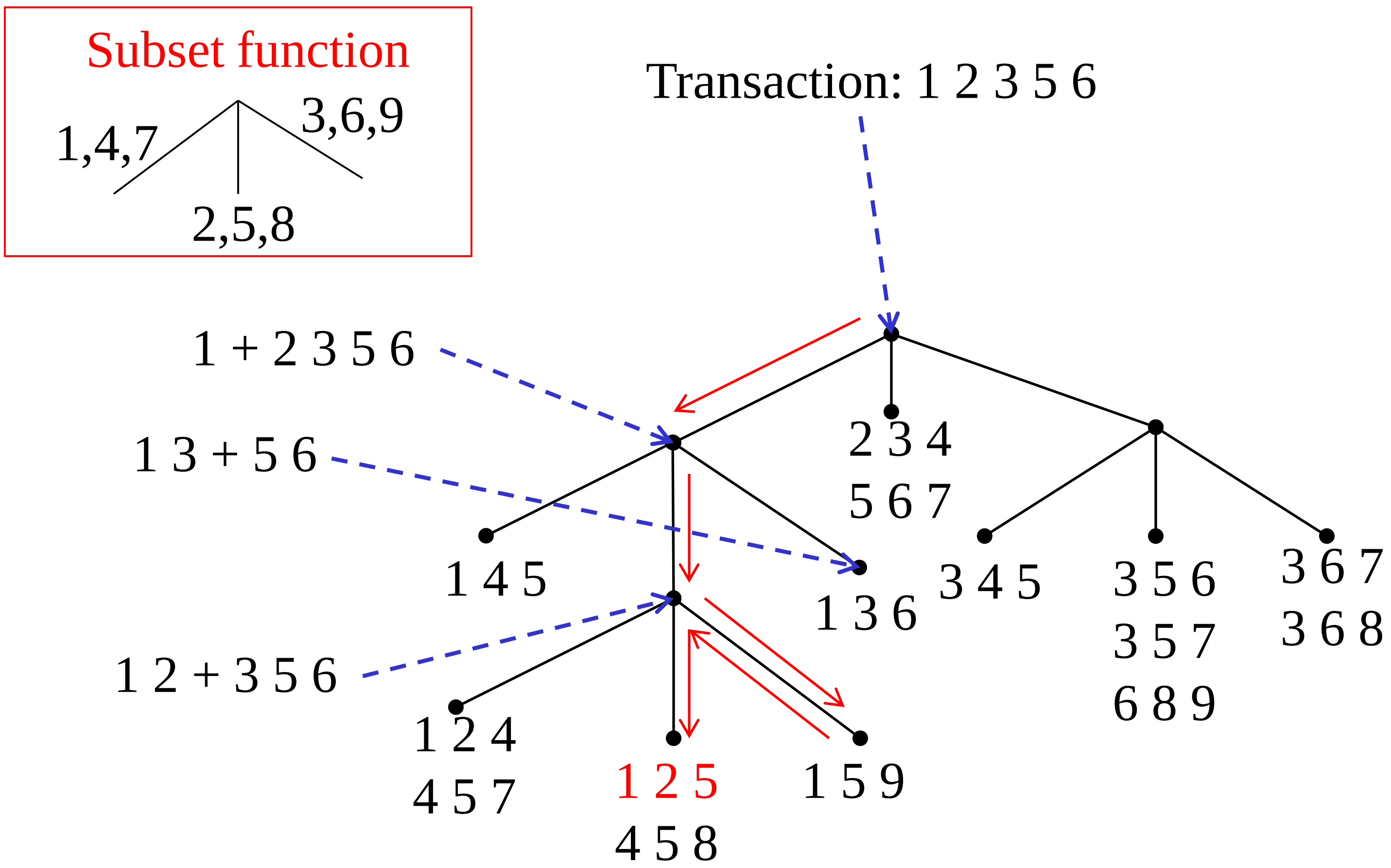
  forall ***(k-1)-subsets s of c*** do

  if *(s is not in* L_{k-1}*)* **then delete** *c* **from** C_k

How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a *hash-tree*
  - *Leaf* node of hash-tree contains a list of itemsets and counts
  - *Interior* node contains a hash table
  - *Subset function*: finds all the candidates contained in a transaction

Example: Counting Supports of Candidates



Efficient Implementation of Apriori in SQL

- Hard to get good performance out of pure SQL (SQL-92) based approaches alone
- Make use of object-relational extensions like UDFs, BLOBs, Table functions etc.
  - Get orders of magnitude improvement
- S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In SIGMOD'98

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

2. Hash-based Technique

- A hash-based technique can be used to reduce the candidate k-itemsets C_k for k>1.
- To generate the frequent 1-itemsets, L1
- Generate all the 2-itemsets for each transaction, hash them into the different buckets of a hash table and increase the bucket count .
- If the bucket count is less than the threshold value is not frequent.

Hash-based Technique (Contd.)

Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Hash-based Technique (Contd.)

$$h(x, y) = \text{order}(x) \times 10 + \text{order}(y) \bmod 7 \quad [0 \text{ to } 6]$$

T_{100}

$$h(I_1, I_2) = 1 \times 10 + 2 \bmod 7$$
$$= 12 \bmod 7 = 5$$

$$h(I_1, I_5) = 1 \times 10 + 5 \bmod 7$$
$$= 15 \bmod 7 = 1$$

$$h(I_2, I_5) = 2 \times 10 + 5 \bmod 7$$
$$= 20 + 5 \bmod 7 = 4$$

Hash-based Technique (Contd.)

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

2. Transaction Reduction

- A transaction that does not contain any frequent *k*-itemsets cannot contain any frequent (*k* +1)-itemsets.
- That transaction can be marked or removed
- Subsequent database scans for j- itemsets, (*j* > *k)*, will need not to consider.

Partition: Scan Database Only Twice

- A partitioning technique can be used that requires just two database scans to mine the frequent itemsets.

- **Phase I:** Divides the transactions of *D* into *n* nonoverlapping partitions.

- If the minimum relative support threshold for transactions in *D* is *min_sup*, then the minimum support count for a partition is
  - *min_sup X the number of transactions in that partition*.
- For each partition, all the *local frequent itemsets* (i.e., the itemsets frequent within the partition) are found.

Partition: Scan Database Only Twice

- A local frequent itemset may or may not be frequent with respect to the entire database, *D*.

- *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions*.

- All local frequent itemsets are candidate itemsets with respect to *D*.

- The collection of frequent itemsets from all partitions forms the *global candidate itemsets* with respect to *D*.

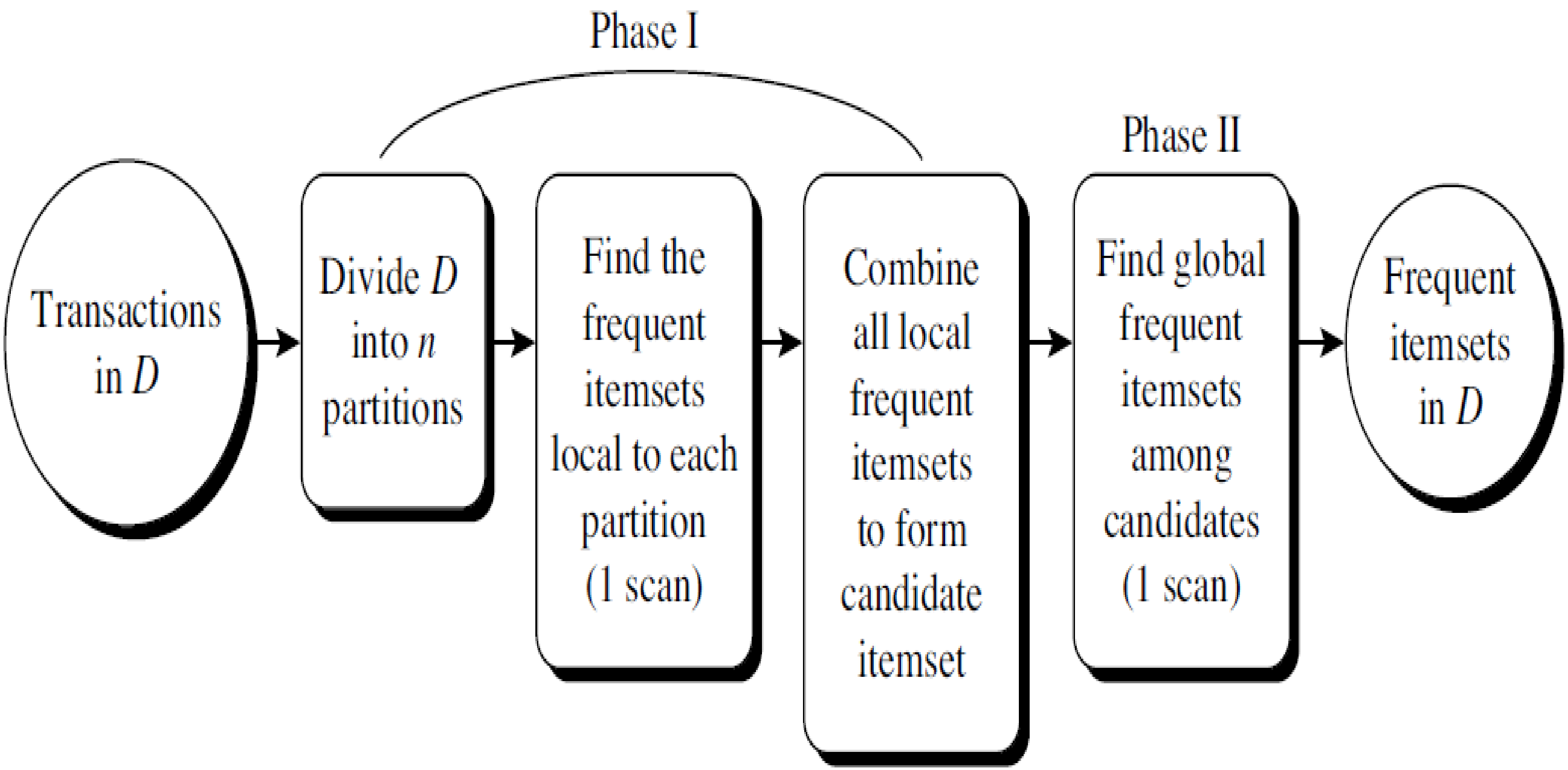
Partition: Scan Database Only Twice

- **Phase II:**
- a second scan of *D* is conducted

- The actual support of each candidate is assessed to determine the global frequent itemsets.

- Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Partition: Scan Database Only Twice



Mining by partitioning the data

Dynamic itemset counting

- The database is partitioned into blocks marked by start points.
- In this variation, new candidate itemsets can be added at any start point,
- Unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.
- The technique uses the count-so-far as the lower bound of the actual count.

Dynamic itemset counting

- If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates.

- This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

DHP: Reduce the Number of Candidates

- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} …
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*

Problems with Apriori/Apriori Extensions

- Many cases the the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance.

- It can suffer from
- Still need to generate a huge number of candidate sets.
  - if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.
- It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching.
- It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

"Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?"

Frequent pattern growth or FP-Growth

- finding frequent itemsets without candidate generation.
- It adopts a divide-and-conquer strategy
- First,it compresses the database representing frequent items into a frequent pattern tree.
- It then divides the compressed database into a set of conditional databases.

FP-Growth

- Step 1. Scan DB once, find frequent 1-itemsets (single items)
- Step 2. Order frequent items in frequency descending order
- Step 3. Scan DB again, construct FP-tree

FP-Growth: Example

Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

FP-Growth: Example

- Frequent 1-itemsets
- Let the minimum support count =2.
- The set of frequent items is sorted in the order of descending support count.
- The resultant set or list is denoted by L.
- L = {{I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, {I5: 2}}.

FP-Growth: Example

- FP-Tree Construction
- Step1: First, create the root of the tree, labeled with "null."
- Step2: Scan the database D a second time.
  - The items in each transaction are processed in **L** order.
- Step3: a branch (path) is created for each transaction.

FP-Growth: Example

- The scan of the first transaction, “T100: I1, I2, I5,” which contains three items (I2, I1, I5 in L order),
- leads to the construction of the first branch of the tree with three nodes, <I2: 1>, <I1: 1>, and <I5: 1>,
- I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1.

- The second transaction, T200, contains the items I2 and I4 in L order,
- A branch where I2 is linked to the root and I4 is linked to I2.
  - this branch would share a common prefix, I2, with the existing path for T100.

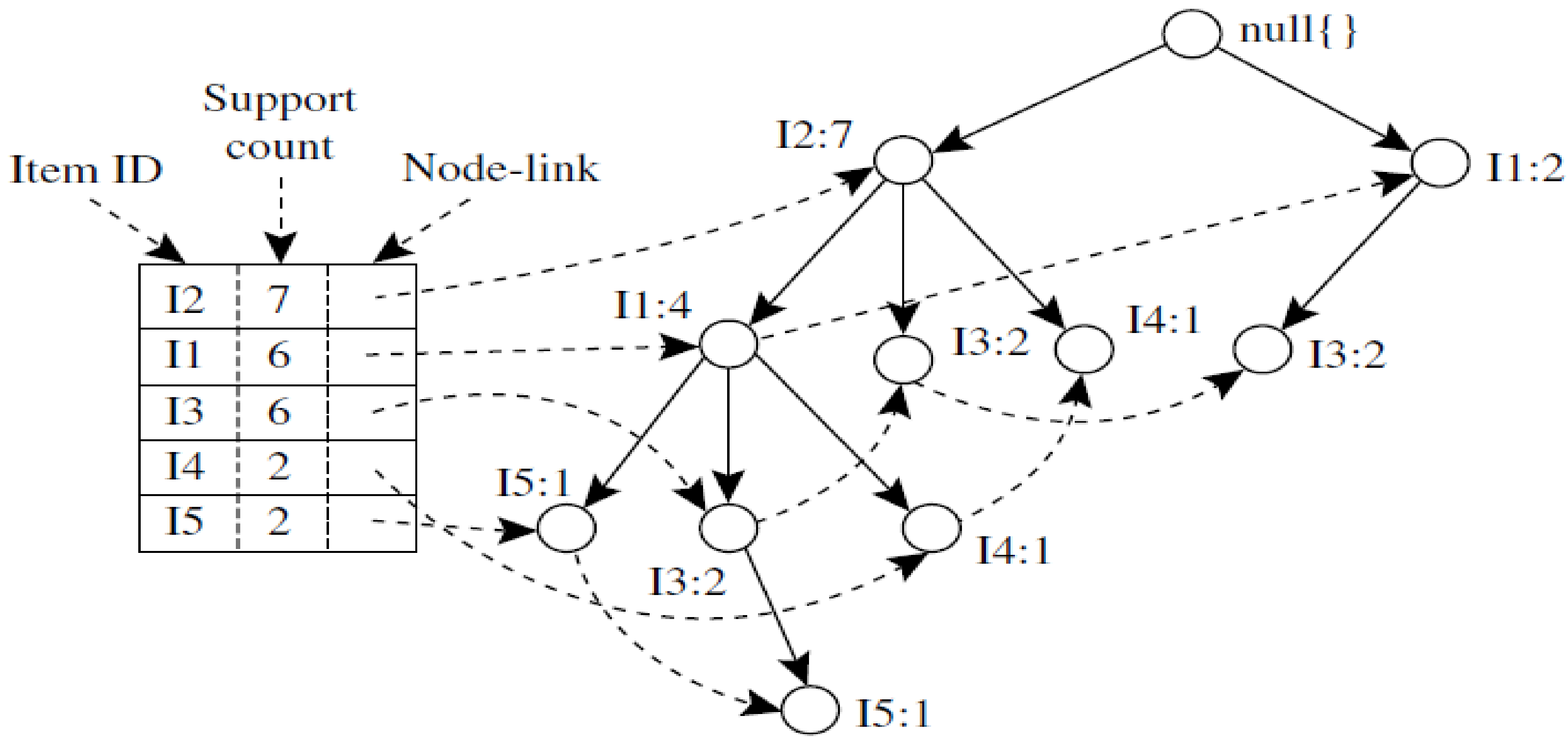
FP-Growth: Example

- When considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1.

- The nodes for the items following the prefix are created and linked accordingly.

- **Header table:**
  - It facilitate tree traversal,
  - an item header table is built
  - each item points to its occurrences in the tree via a chain of node-links

FP-Growth: Example



FP-tree

Mining FP-Growth

- Start from each frequent length-1 pattern (as an initial suffix pattern)

- Construct its conditional pattern base (a “sub-database,” which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern),

- Construct its (conditional) FP-tree, and perform mining recursively on the tree.

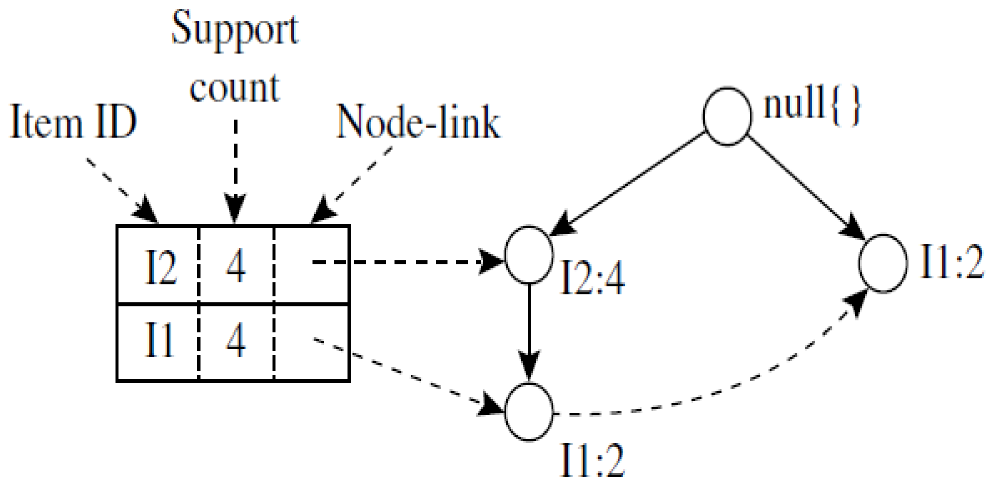
- The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Conditional Pattern Base

Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Conditional Pattern Base



Vertical Data format

- Apriori and FP-grothTID-itemset format (i.e., {TID : itemset}), where TID is a transaction ID and itemset is the set of items ,it is known as the horizontal data format.

- item-TID set format (i.e., {item : TID set}), where item is an item name, and TID set is the set of transaction identifiers containing the item. This is known as the vertical data format

Mining frequent itemsets using the vertical data format

- Consider the horizontal data format of the transaction database, D,

The Vertical Data Format of the Transaction Data Set *D* of Table 6.1

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

- It is transformed into the vertical data format by scanning the data set once

- Mining can be performed on this data set by intersecting the TID sets of every pair of frequent single items.

Mining frequent itemsets using the vertical data format

- The minimum support count is 2
- 10 intersections performed in total, which lead to eight nonempty 2-itemsets

2-Itemsets in Vertical Data Format

itemset	*TID_set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Mining frequent itemsets using the vertical data format: Process

- First, we transform the horizontally formatted data into the vertical format by scanning the data set once.
- The support count of an itemset is simply the length of the TID set of the itemset.
- Starting with k = 1, the frequent k-itemsets can be used to construct the candidate (k + 1)-itemsets based on the Apriori property.
- The computation is done by intersection of the TID sets of the frequent k-itemsets to compute the TID sets of the corresponding (k + 1)-itemsets.
- This process repeats, with k incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Closed and maximal frequent itemsets.

- Suppose that a transaction database has only two transactions: {ha1, a2,..., a100i; ha1, a2,..., a50i}.

- Let the minimum support count threshold be min sup = 1.

- We find two closed frequent itemsets and their support counts, that is, C = {{a1, a2,..., a100} : 1; {a1, a2,..., a50} : 2}.

- There is only one maximal frequent itemset: M = {{a1, a2,..., a100} : 1}.

- We cannot include {a1, a2,..., a50} as a maximal frequent itemset because it has a frequent superset, {a1, a2,..., a100}.

- Compare this to the preceding where we determined that there are 2 100 – 1 frequent itemsets, which are too many to be enumerated!

- The set of closed frequent itemsets contains complete information regarding the frequent itemsets.

- For example, from C, we can derive, say, (1) {a2, a45 : 2} since {a2, a45} is a sub-itemset of the itemset {a1, a2,..., a50 : 2}; and (2) {a8, a55 : 1} since {a8, a55} is not a sub-itemset of the previous itemset but of the itemset {a1, a2,..., a100 : 1}.

- However, from the maximal frequent itemset, we can only assert that both itemsets ({a2, a45} and {a8, a55}) are frequent, but we cannot assert their actual support counts.

Chapter 5: Mining Frequent Patterns, Association and Correlations

- Basic concepts and a road map
- Efficient and scalable frequent itemset mining methods
- Mining various kinds of association rules
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

Mining Various Kinds of Association Rules

- Mining Association Rules
- Mining multilevel association
- Miming multidimensional association
- Mining quantitative association
- Mining interesting correlation patterns

Association Rule Mining

- Let I = {I1, I2,..., Im} be an itemset.
- Let D, the task-relevant data, be a set of database transactions where each transaction T is a nonempty itemset such that $T \subseteq I$.
- Each transaction is associated with an identifier, called a TID.

- Let A be a set of items.
- A transaction T is said to contain A if $A \subseteq T$.
- An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, A 6= ∅, B 6= ∅, and $A \cap B = \varphi$.

Association Rule Mining

- Association rule mining can be viewed as a two-step process:

- 1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min sup.

- 2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

Association Rule Mining

- The rule $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ holds in the transaction set **D** with support ***s***, where ***s*** is the percentage of transactions in **D** that contain A ∪ B (i.e., the union of sets A and B say, or, both A and B), i.e the probability, P(A ∪B).

- The rule A ⇒ B has confidence **c** in the transaction set **D**, where **c** is the percentage of transactions in **D** containing **A** that also contain **B**.

- Note: Thresholds can be a set by users or domain experts.

Association Rule Mining

- This is taken to be the conditional probability, $P(B|A)$.

$$support(A \Rightarrow B) = P(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B|A).$$

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}$$

Association Rule Mining

- Rules that satisfy both a minimum support threshold (min sup) and a minimum confidence threshold (min conf) are called strong.

- We write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0

Association Rule Mining

- Note that the itemset support referred to as relative support, whereas the occurrence frequency is called the absolute support.

- If the relative support of an itemset I satisfies a prespecified minimum support threshold (i.e., the absolute support of I satisfies the corresponding minimum support count threshold), then I is a frequent itemset.
- The set of frequent k-itemsets is commonly denoted by L_k.

Association Rule Mining

- A set of items is referred to as an itemset.
- An itemset that contains k items is a k-itemset. The set {computer, antivirus software} is a 2-itemset.
- The occurrence frequency of an itemset is the number of transactions that contain the itemset.
- This is also known, simply, as the frequency, support count, or count of the itemset.

Closed and maximal frequent itemsets

- Suppose that a transaction database has only two transactions:
  - {<a1, a2,..., a100>; <a1, a2,..., a50>}.

- Let the minimum support count threshold be min sup = 1.

- We find two closed frequent itemsets and their support counts, that is, C = {{a1, a2,..., a100} : 1; {a1, a2,..., a50} : 2}.

- There is only one maximal frequent itemset: M = {{a1, a2,..., a100} : 1}.

Association Rule Mining

- We cannot include {a1, a2,..., a50} as a maximal frequent itemset because it has a frequent superset, {a1, a2,..., a100}.

Mining Multiple-Level Association Rules

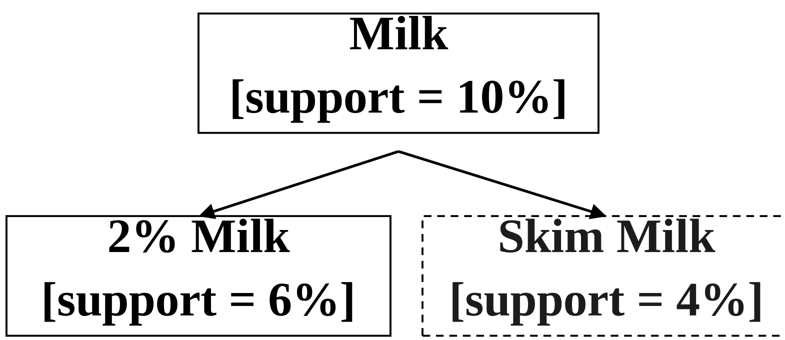
- Items often form hierarchies
- Flexible support settings
  - Items at the lower level are expected to have lower support
- Exploration of *shared* multi-level mining (Agrawal & Srikant@VLB'95, Han & Fu@VLDB'95)

uniform support

reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 5%



Level 1
min_sup = 5%

Level 2
min_sup = 3%

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to "ancestor" relationships between items.
- Example
  - milk $\Rightarrow$ wheat bread [support = 8%, confidence = 70%]
  - 2% milk $\Rightarrow$ wheat bread [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the "expected" value, based on the rule's ancestor.

Handling Multiple Constraints

- Different constraints may require different or even conflicting item-ordering
- If there exists an order *R* s.t. both C_1 and C_2 are convertible w.r.t. *R*, then there is no conflict between the two convertible constraints
- If there exists conflict on order of items
  - Try to satisfy one constraint first
  - Then using the order for the other constraint to mine frequent itemsets in the corresponding projected database

What Constraints Are Convertible?

Constraint	Convertible anti-monotone	Convertible monotone	Strongly convertible
avg(S) $\leq$, $\geq$ v	Yes	Yes	Yes
median(S) $\leq$, $\geq$ v	Yes	Yes	Yes
sum(S) $\leq$ v (items could be of any value, v $\geq$ 0)	Yes	No	No
sum(S) $\leq$ v (items could be of any value, v $\leq$ 0)	No	Yes	No
sum(S) $\geq$ v (items could be of any value, v $\geq$ 0)	No	Yes	No
sum(S) $\geq$ v (items could be of any value, v $\leq$ 0)	Yes	No	No
……			

Chapter 6. Classification and Prediction

- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification by decision tree induction
- Bayesian classification
- Rule-based classification
- Classification by back propagation
- Support Vector Machines (SVM)
- Associative classification
- Lazy learners (or learning from your neighbors)
- Other classification methods
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Classification vs. Prediction

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit approval
  - Target marketing
  - Medical diagnosis
  - Fraud detection

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

Process (1): Model Construction

Training Data

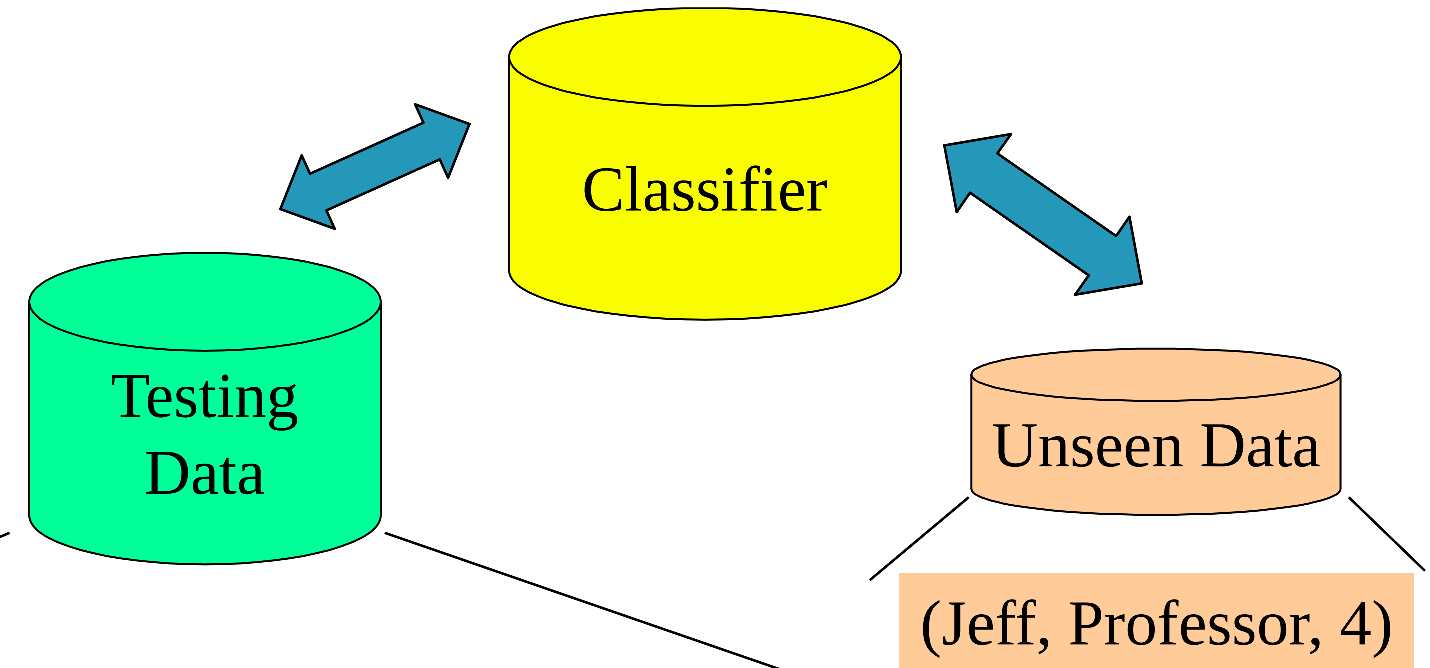
Classification Algorithms

Classifier (Model)

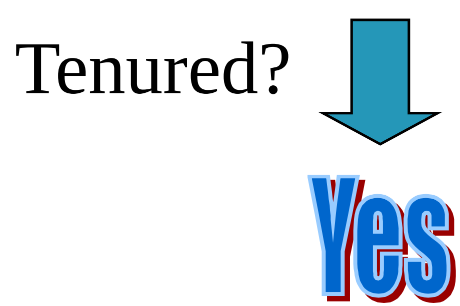
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = 'professor'
OR years > 6
THEN tenured = 'yes'

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes



Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

Chapter 6. Classification and Prediction

- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification by decision tree induction
- Bayesian classification
- Rule-based classification
- Classification by back propagation
- Support Vector Machines (SVM)
- Associative classification
- Lazy learners (or learning from your neighbors)
- Other classification methods
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Issues: Data Preparation

- Data cleaning
  - Preprocess data in order to reduce noise and handle missing values
- Relevance analysis (feature selection)
  - Remove the irrelevant or redundant attributes
- Data transformation
  - Generalize and/or normalize data

Issues: Evaluating Classification Methods

- Accuracy
  - classifier accuracy: predicting class label
  - predictor accuracy: guessing value of predicted attributes
- Speed
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Chapter 6. Classification and Prediction

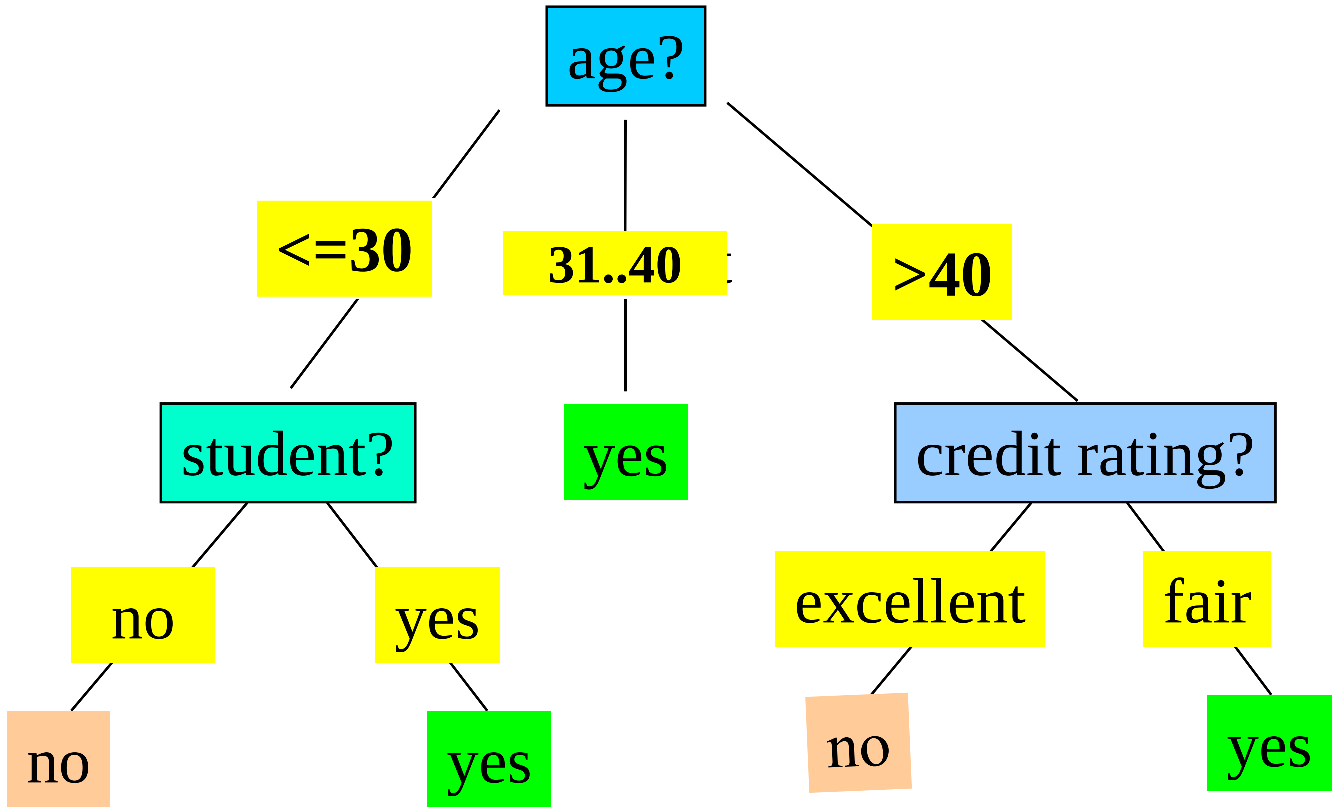
- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification by decision tree induction
- Bayesian classification
- Rule-based classification
- Classification by back propagation
- Support Vector Machines (SVM)
- Associative classification
- Lazy learners (or learning from your neighbors)
- Other classification methods
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Decision Tree Induction: Training Dataset

This follows an example of Quinlan's ID3 (Playing Tennis)

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

Output: A Decision Tree for "*buys_computer*"



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

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"

Enhancements to Basic Decision Tree Induction

- Allow for continuous-valued attributes
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle missing attribute values
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- Attribute construction
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

Classification in Large Databases

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why decision tree induction in data mining?
  - relatively faster learning speed (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use SQL queries for accessing databases
  - comparable classification accuracy with other methods

Chapter 6. Classification and Prediction

- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification by decision tree induction
- Bayesian classification
- Rule-based classification
- Classification by back propagation
- Support Vector Machines (SVM)
- Associative classification
- Lazy learners (or learning from your neighbors)
- Other classification methods
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayesian Theorem: Basics

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|**X**), the probability that the hypothesis holds given the observed data sample **X**
- P(H) (*prior probability*), the initial probability
  - E.g., **X** will buy computer, regardless of age, income, ...
- P(**X**): probability that sample data is observed
- P(**X**|H) (*posteriori probability*), the probability of observing the sample **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayesian Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

    posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_2 iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Towards Naïve Bayesian Classifier

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Derivation of Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i, D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayesian Classifier: Training Dataset

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	_comp
<=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

Naïve Bayesian Classifier: An Example

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Avoiding the 0-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be non-zero. Otherwise, the predicted prob. will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10),
- Use Laplacian correction (or Laplacian estimator)
  - Adding 1 to each case

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayesian Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.

      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayesian Classifier
- How to deal with these dependencies?
  - Bayesian Belief Networks

Chapter 6. Classification and Prediction

- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification by decision tree induction
- Bayesian classification
- Rule-based classification
- Classification by back propagation
- Support Vector Machines (SVM)
- Associative classification
- Lazy learners (or learning from your neighbors)
- Other classification methods
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Using IF-THEN Rules for Classification

- Represent the knowledge in the form of IF-THEN rules

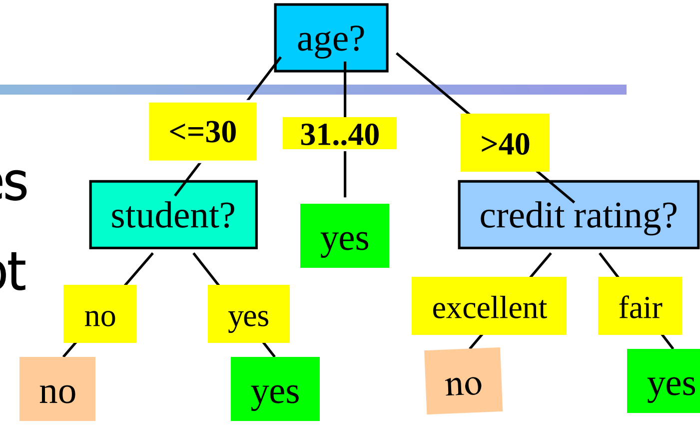
  R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes

  - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

  coverage(R) = n_{covers} /|D| /* D: training data set */

  accuracy(R) = $n_{correct}$ / n_{covers}
- If more than one rule is 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 test*)
  - 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
- Example: Rule extraction from our *buys_computer* decision-tree

IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*

IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*

IF *age* = mid-age THEN *buys_computer* = *yes*

IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*

IF *age* = young AND *credit_rating* = *fair* THEN *buys_computer* = *no*

Rule Extraction from the Training Data

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - The process repeats on the remaining tuples unless *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

How to Learn-One-Rule?

- Star with the most general rule possible: condition = empty
- Adding new attributes by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Rule-Quality measures: consider both coverage and accuracy
  - Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition

$$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg})$$

    It favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

    Pos/neg are # of positive/negative tuples covered by R.

    If *FOIL_Prune* is higher for the pruned version of R, prune R