Association Rule Mining

- Association Rules and Frequent Patterns
- Frequent Pattern Mining Algorithms
  - Apriori
  - FP-growth
- Correlation Analysis
- Constraint-based Mining
- Using Frequent Patterns for Classification
  - Associative Classification (rule-based classification)
  - Frequent Pattern-based Classification

Association Rules

- A Frequent pattern is a pattern (a set of items, subsequences, subgraphs, etc.) that occurs frequently in a data set.
- Motivation: Finding inherent regularities (associations) in data.
- Forms the foundation for many essential data mining tasks:
  - Association, correlation, and causality analysis
  - Classification: associative classification
  - Cluster analysis: frequent pattern-based clustering
  - …
- First proposed by [AIS93] in the context of frequent itemsets and association rule mining for market basket analysis.
- Extended to many different problems: graph mining, sequential pattern mining, times series pattern mining, text mining…

Association Rules

- An item (I) is:
  - For market basket data: I is an item in the store, e.g. milk.
  - For relational data: I is an attribute-value pair (numeric attributes should be discretized), e.g. salary=high, gender=male.
- A *pattern* (P) is a conjunction of items: $P=I_1 \wedge I_2 \wedge \ldots I_n$ (itemset)
- A pattern defines a group (subpopulation) of instances.
- Pattern P' is subpattern of P if $P' \subset P$
- A rule R is $A \Rightarrow B$ where A and B are disjoint patterns.
- Support(A $\Rightarrow$ B)=P(A $\cup$ B)
- Confidence(A $\Rightarrow$ B)=P(B|A)=posterior probability

Association Rules

- Framework: find all the rules that satisfy both *a* minimum support (*min_sup*) and a minimum confidence (*min_conf*) thresholds.

- Association rule mining:
  - Find all frequent patterns (with support ≥ *min_sup*).
  - Generate strong rules from the frequent patterns.

- The second step is straightforward:
  - For each frequent pattern *p*, generate all nonempty subsets.
  - For every non-empty subset *s*, output the rule $s \Rightarrow (p\text{-}s)$ if conf=sup(p)/sup(s) ≥ *min_conf*.

- The first step is much more difficult. Hence, we focus on frequent pattern mining.

Association Rules
Example for market basket data

- Items={A,B,C,D,E,F}

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

Let min_sup = 60% (3)

min_conf = 50%

FP= {A:3, B:3, D:4, E:3, AD:3}

Association rules:

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

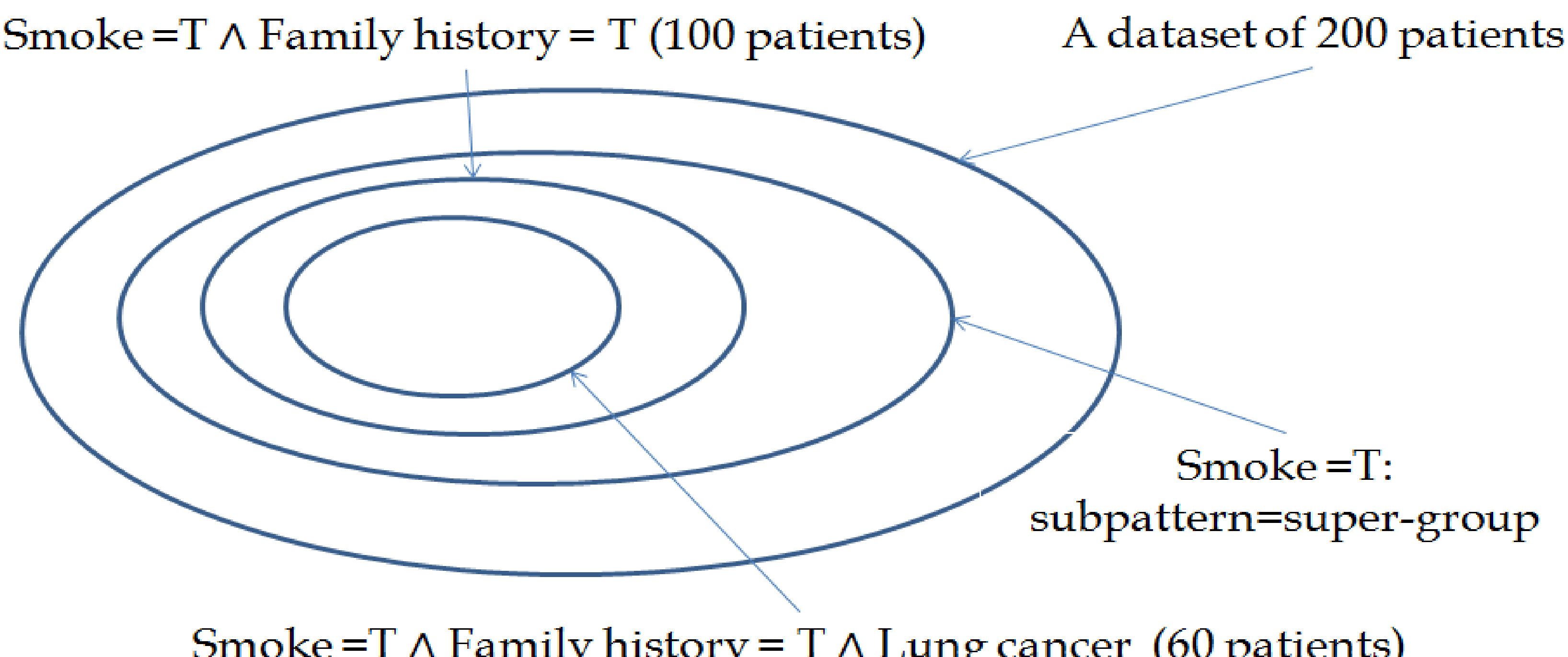
$D \rightarrow A$ (60%, 75%)

Association Rules
Example for relational data

Rule: Smoke =T ∧ Family history = T ⇒ Lung cancer=T

sup (Smoke =T ∧ Family history = T ∧ Lung cancer=T)= 60/200=30%

conf (Smoke =T ∧ Family history = T ⇒ Lung cancer=T)= 60/100=60%



Frequent Pattern Mining

- Scalable mining methods: Three major approaches:
  - Apriori [Agrawal & Srikant 1994]
  - Frequent pattern growth (FP-growth) [Han, Pei & Yin 2000]
  - Vertical data format approach [Zaki 2000]

Apriori

- The Apriori property:
  - Any subset of a frequent pattern must be frequent.
  - If {beer, chips, nuts} is frequent, so is {beer, chips}, i.e., every transaction having {beer, chips, nuts} also contains {beer, chips}.

- <u>Apriori pruning principle</u>: If there is any pattern which is infrequent, its superset should not be generated/tested!

- Method (level-wise search):
  - Initially, scan DB once to get frequent 1-itemset
  - For each level k:
    - Generate length (k+1) candidates from length k frequent patterns
    - Scan DB and remove the infrequent candidates
  - Terminate when no candidate set can be generated

Apriori

min_sup = 2

Database

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

Apriori

- Candidate generation: Assume we are generating k+1 candidates at level k
  - Step 1: *self-joining* two frequent k-patterns if they have the same k-1 prefix
  - Step 2: *pruning*: remove a candidate if it contains any infrequent k-pattern.
- Example: L_3={*abc*, *abd*, *acd*, *ace*, *bcd*}
  - Self-joining: $L_3 * L_3$
    - *abc* and *abd* ⇒ *abcd*
    - *acd* and *ace* ⇒ *acde*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

Apriori

- The bottleneck of *Apriori*:
  - Huge candidate sets:
    - To discover a frequent 100-pattern, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 10^{30}$ candidates!
  - Multiple scans of database:
    - Needs $(n+1)$ scans, n is the length of the longest pattern.
- Can we avoid candidate generation?

FP-growth

- The FP-growth algorithm: mining frequent patterns without candidate generation [Han, Pei & Yin 2000]
- Compress a large database into a compact Frequent-Pattern tree (FP-tree) structure
  – highly condensed, but complete for frequent pattern mining
  – avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  – A divide-and-conquer methodology: decompose mining tasks into smaller ones
  – Avoid candidate generation: sub-database test only!

FP-growth
Constructing the FP-tree

TID	Items bought	(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}	{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}

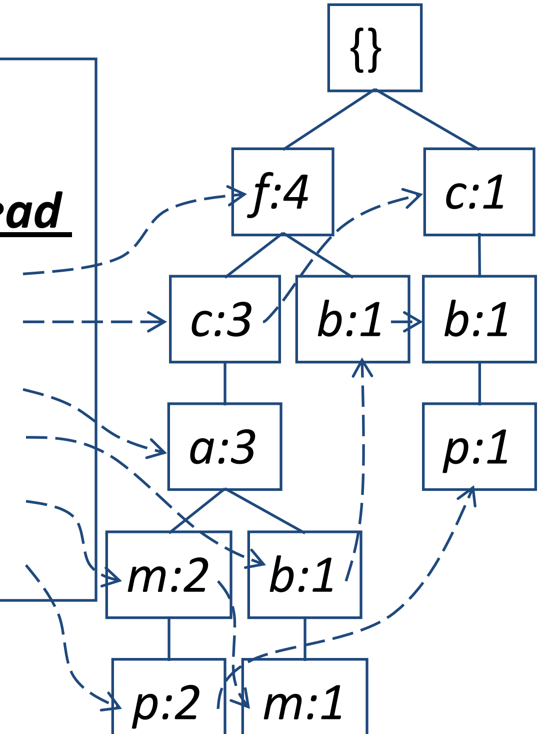
Steps:

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

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

min_sup = 3



FP-growth

- Method (divide-and-conquer)
  - For each item, construct its conditional pattern-base, and then its conditional FP-tree.
  - Repeat the process on each newly created conditional FP-tree.
  - Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

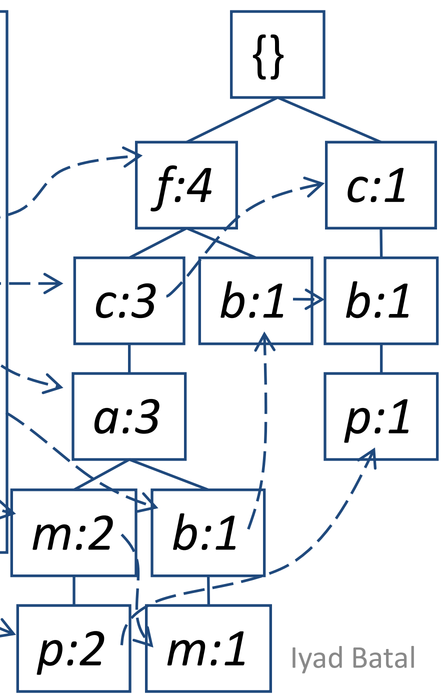
FP-growth

Step 1: From FP-tree to Conditional Pattern Base

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



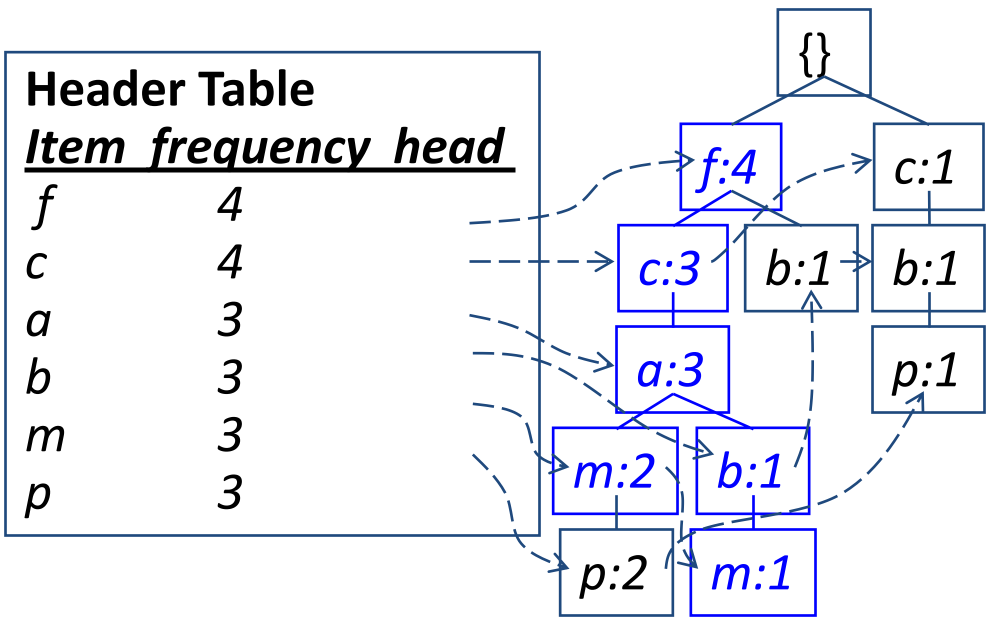
Conditional **pattern bases**

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

FP-growth

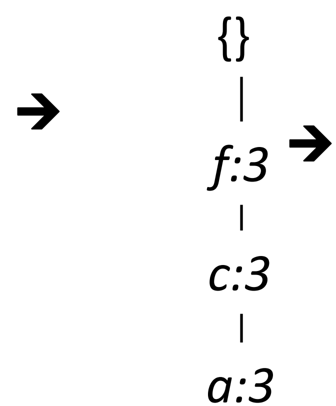
Step 2: Construct Conditional FP-tree

- Start from the end of the list
- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base
- Example: Here we are mining for pattern m, min_sup=3



m-conditional **pattern base:**

fca:2, fcab:1



All frequent patterns concerning *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

m-conditional **FP-tree**

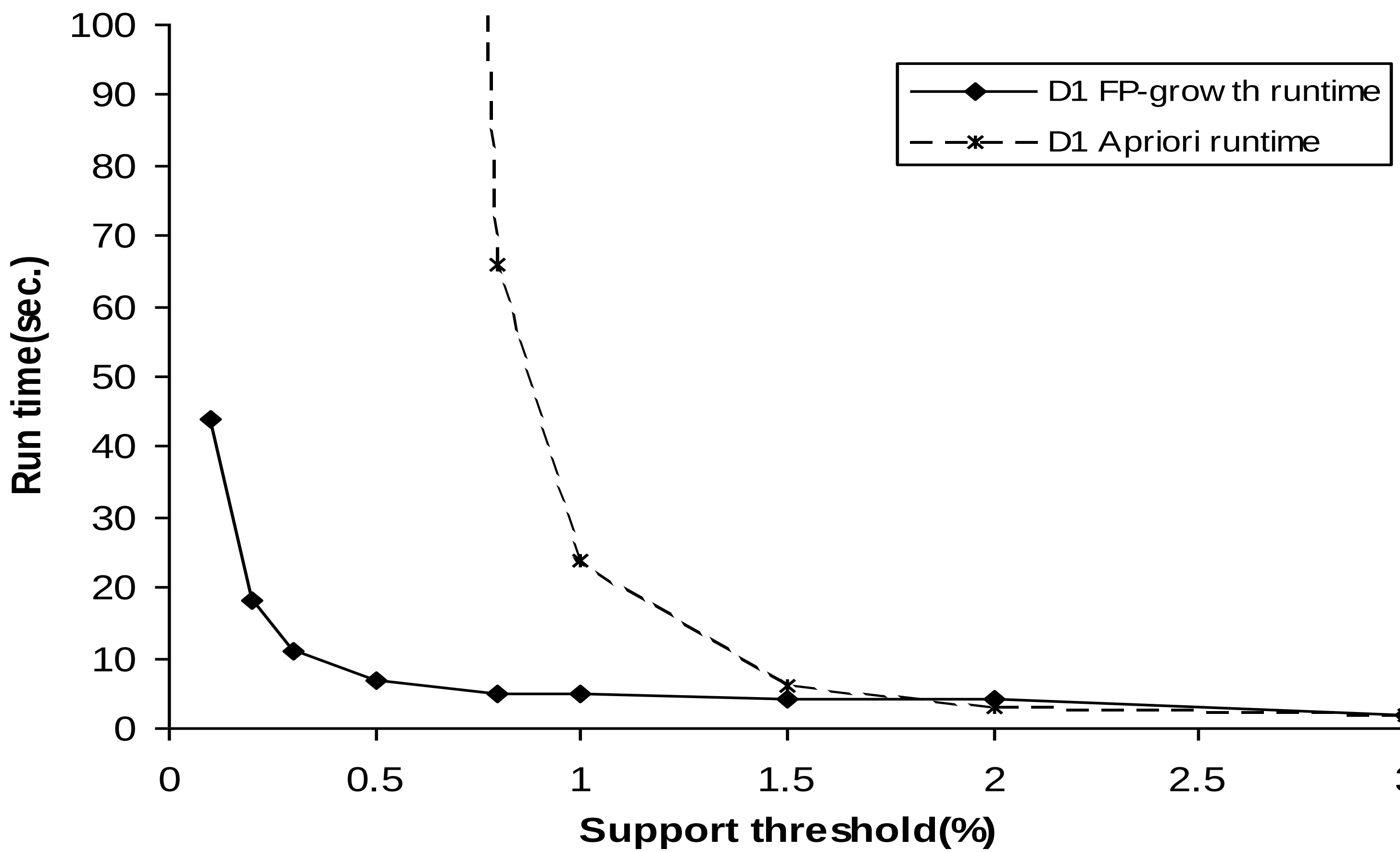
FP-growth

- FP-growth is faster than Apriori because:
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building (no pattern matching)

- Disadvantage: FP-tree may not fit in main memory!

FP-growth

FP-growth vs. Apriori: Scalability With the Support Threshold



Correlation analysis

- Association rule mining often generates a huge number of rules, but a majority of them either are redundant or do not reflect the true correlation relationship among data objects.

- Some strong association rules (based on support and confidence) can be misleading.

- Correlation analysis can reveal which strong association rules are interesting and useful.

Correlation analysis

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence

Contingency table

	Basketball	Not basketball	Sum (row)
Cereal	2000 (40%)	1750 (35%)	3750 (75%)
Not cereal	1000 (20%)	250 (5%)	1250 (25%)
Sum(col.)	3000 (60%)	2000 (40%)	5000 (100%)

Correlation analysis
The lift score

$$lift(A \Rightarrow B) = \frac{P(A \cup B)}{P(A)P(B)} = \frac{P(B \mid A)}{P(B)}$$

- Lift = 1 $\Rightarrow$ A and B are independent
- Lift > 1 $\Rightarrow$ A and B are positively correlated
- Lift < 1 $\Rightarrow$ A and B are negatively correlated.

	Basketball	Not basketball	Sum (row)
Cereal	2000 (40%)	1750 (35%)	3750 (75%)
Not cereal	1000 (20%)	250 (5%)	1250 (25%)
Sum(col.)	3000 (60%)	2000 (40%)	5000 (100%)

$$lift(\text{basketball} \Rightarrow cereal) = \frac{2000/5000}{3000/5000 * 3750/5000} = 0.89$$

$$lift(basketball \Rightarrow \neg cereal) = \frac{1000/5000}{3000/5000 * 1250/5000} = 1.33$$

Correlation analysis
The χ2 test

- Lift calculates the correlation value, but we could not tell whether the value is statistically significant.

- Pearson *Chi-square* is the most common test for significance of the relationship between categorical variables

$$x^2 = \sum \frac{(O(r) - E[r])^2}{E[r]}$$

- If this value is larger than a cutoff value at a significance level (e.g. at 95% significance level), then we say all the variables are dependent (correlated), else we say all the variables are independent.

Other correlation/interestingness measure: cosine, all confidence, IG…

Correlation analysis disadvantages

Problem: Evaluate each rule individually!

Pr(CHD)=30%

R2: Family history=yes ∧ Race=Caucasian ⇒ CHD

[sup=20%, conf=55%]

R2 is interesting!

R1: Family history=yes ⇒ CHD

[sup=50%, conf=60%]

R2 is not interesting!

We should consider the nested structure of the rules!

To solve this problem, we proposed the MDR framework.

Constraint-based Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface).
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
    - Specify the task relevant data, the relevant attributes, rule templates, additional constraints…
  - System optimization: explores such constraints for efficient mining—constraint-based mining.

Constraint-based Mining

- Anti-monotonic constraints are very important because they can greatly speed up the mining process.
- Anti-monotonicity exhibit an Apriori-like property:
  - When a pattern **violates** the constraint, so does any of its superset
  - sum(S.Price) $\leq$ v is anti-monotone
  - sum(S.Price) $\geq$ v is not anti-monotone
- Some constraints can be converted into anti-monotone constraints by properly ordering items
  - Example : avg(S.profit) $\geq$ 25
    - Order items in value-descending order, it becomes anit-monotone!

Association Rule Mining

- Association Rules and Frequent Patterns
- Frequent Pattern Mining Algorithms
  - Apriori
  - FP-growth
- Correlation Analysis
- Constraint-based Mining
- Using Frequent Patterns for Classification
  - Associative Classification (rule-based classification)
  - Frequent Pattern-based Classification

Associative classification

- Associative classification: build a rule-based classifier from association rules.
- This approach overcomes some limitations of greedy methods (e.g. decision-tree, sequential covering algorithms), which considers only one attribute at a time (found to be more accurate than C4.5).
- Build class association rules:
  - Association rules in general can have any number of items in the consequent.
  - Class association rules set the consequent to be the class label.
- Example: Age=youth $\wedge$ Credit=OK $\Rightarrow$ buys_computer=yes

  [sup=20%, conf=90%]

Associative classification
CBA

- CBA: Classification-Based Association [Liu et al, 1998]
- Use the Apriori algorithm to mine the class association rules.
- Classification:
  - Organize the rules according to their confidence and support.
  - classify a new example x by the first rule satisfying x.
  - Contains a default rule (with lowest precedence).

Associative classification CMAR

- CMAR (Classification based on Multiple Association rules) [Li et al 2001]
- Use the FP-growth algorithm to mine the class association rules.
- Employs the CR_tree structure (prefix tree for indexing the rules) to efficiently store and retrieve rules.
- Apply rule pruning whenever a rule is inserted in the tree:
  - If R1 is more general than R2 and conf(R1)>conf(R2): R2 is pruned
  - All rules for which the antecedent and class are not positively correlated ($\chi 2$ test) are also pruned.
- CMAR considers multiple rules when classifying an instance and use a weighted measure to find the strongest class.
- CMAR is slightly more accurate and more efficient than CBA

Associative classification
Harmony

- Drawback of CBA and CMAR is that the number of rules can be extremely large.
- Harmony [Wang et al, 2005] adopts an instance-centric approach:
  - Find the highest confidence rule for each training instance.
  - Build the classification model from the union of these rules.
- Use the FP-growth algorithm to mine the rules.
- Efficient mining:
  - Naïve way: mine all frequent patterns and then extract the highest confidence rule for each instance.
  - Harmony employs efficient pruning methods to accelerate the rule discovery: the pruning methods are incorporated within the FP-growth algorithm.

Frequent pattern-based classification

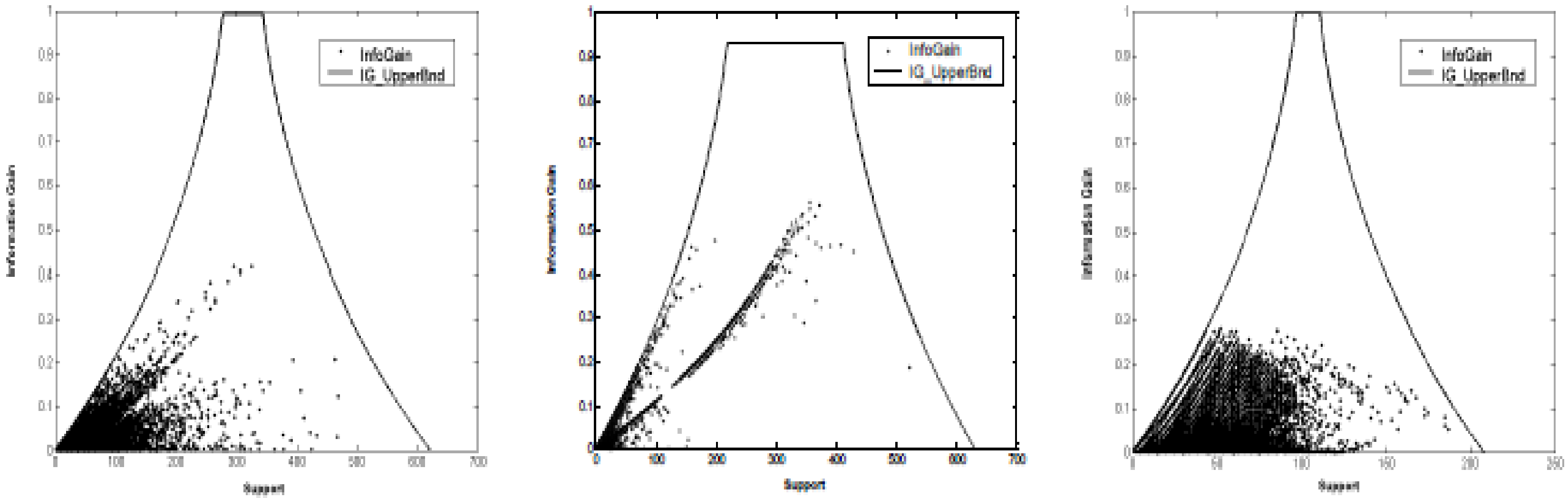
- The classification model is built in the feature space of single features as well as frequent patterns, i.e. map the data to a higher dimensional space.
- Feature combination can capture more underlying semantics than single features.
- Example: word phrases can improve the accuracy of document classification.
- FP-based classification been applied to many problems:
  - Graph classification
  - Time series classification
  - Protein classification
  - Text classification

Frequent pattern-based classification

- Naïve solution: given a dataset with n items (attribute-value), enumerate all 2^n items and use them for classification.
- Problems:
  - Computationally infeasible.
  - Overfitting the classifier.
- Solution: use only frequent patterns. Why?
- [Cheng et al. 2007] showed that:
  - low-support features are not very useful for classification.
  - The discriminative power of a pattern is closely related to its support.
  - They derived an upper bound for information gain as a function of the support.

Frequent pattern-based classification

IG upper bound as a function of the support



- The discriminative power of low-support patterns is bounded by a small value.
- The discriminative power of high-support patterns is bounded by a small value (e.g. stop words in text classification).

Frequent pattern-based classification

- The upper bound allows to automatically set *min_sup* based on an IG threshold (IG_0):

$$min\_sup = \theta^* = arg \max_{\theta} \left(IG_{ub}(\theta) \leq IG_0 \right)$$

- Features with support Θ lower than *min_sup* can be skipped:

$$IG(\theta) \leq IG_{ub}(\theta) \leq IG_{ub}(\theta^*) \leq IG_0$$

- [Cheng et al 2007] Adapt the two phases approach:
  - Mine frequent patterns for each class label.
  - Select discriminative features (univariate).
- Feature space includes all the single features as well as the selected (discriminative) frequent patterns.
- Apply the classifier (e.g. SVM or C4.5) in the new feature space.

Tree-based frequent patterns

- [Fan et al. 2008] proposed building a decision tree in the space of frequent patterns as an alternative for the two phases approach [Cheng et al. 2007].

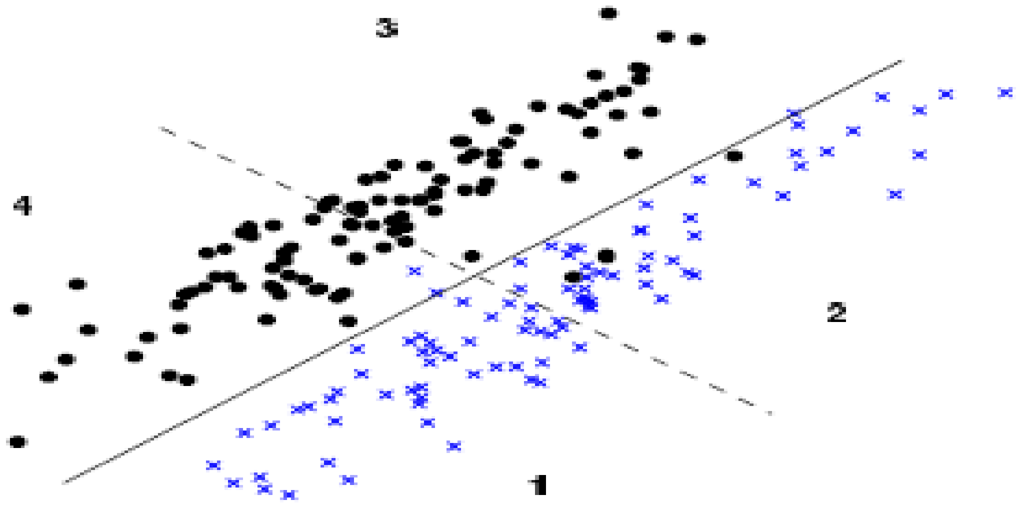
- Arguments against the two phases approach:

1. The number of frequent patterns can be too large for effective feature selection in the second phase.

2. It is difficult to set the optimal *min_sup* threshold:
   - low *min_sup* generate too many candidates and is very slow.
   - High *min_sup* may miss some important patterns.

Tree-based frequent patterns

3. The discriminative power of each pattern is evaluated against the complete dataset (second phase), but not on subset of examples that the other chosen patterns fail to predict well.



After choosing the solid line, the dashed line makes the groups purer (cannot be chosen by the batch mode)

Tree-based frequent patterns

- Method: Construct a decision tree using frequent patterns:
  - Apply frequent pattern on the whole data
  - Select the best frequent pattern *P* to divide the data into two sets: one containing *P* and the other not.
  - Repeat the procedure on each subset until a stopping condition is satisfied.
- Decreasing the support with smaller partitions makes the algorithm able to mine patterns with very low global support.