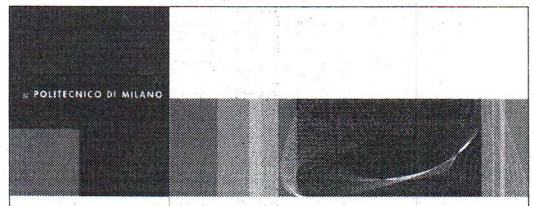
POLITECNICO DI MILANO

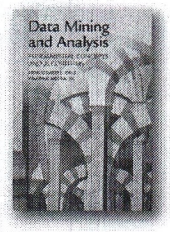


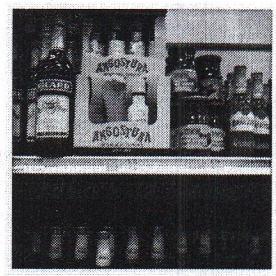
Association Rules

Data Mining and Text Mining

Study Material

- "Data Mining and Analysis" by Zaki & Meira
  - Chapter 8
  - Section 9.1
  - Chapter 10 up to Section 10.2.1 included
  - Section 12.1
- http://www.dataminingbook.info





Market-Basket Transactions

Bread Peanuts Milk Fruit Jam	Bread Jam Soda Chips Milk Fruit	Steak Jam Soda Chips Bread	Jam Soda Peanuts Milk Fruit
Jam Soda Chips Milk Bread	Fruit Soda Chips Milk	Fruit Soda Peanuts Milk	Fruit Peanuts Cheese Yogurt

What is Association Rule Mining?

- Finding frequent patterns and associations among sets of items in transaction databases, relational databases, or other information repositories
- Applications
  - Basket data analysis
  - Cross-marketing
  - Catalog design
  - ...

What is Association Rule Mining?

6

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in a transaction
- Consider the following data set with 8 transactions
- Examples of itemsets
  - {Bread, Milk}
  - {Soda, Chips}
- Examples of association rules
  - {Bread} ⇒ {Milk}
  - {Soda} ⇒ {Chips}
  - {Bread} ⇒ {Jam}

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

We see that bread and milk are often together. Maybe there is an association rule that says "if you have one you also have (probably) the other"

most frequent items

"Itemsets" are the fundamental pattern in association rule mining

What is An Association Rule?

8

this is what we would like to find

- Implication of the form $X \Rightarrow Y$, where X and Y are itemsets
- Example, {Bread} ⇒ {Milk}
- Two rule evaluation metrics
  - Support
  - Confidence

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

evaluation metrics for association rules

{Bread} ⇒ {Milk}

Support

Fraction of transactions that contain both X and Y = how often bread and milk happen together "$P(Y,X)$"

$$s = \frac{\sigma(\{\text{Bread, Milk}\})}{\#\text{ of transactions}} = 0.38$$

Confidence

Measures how often items in Y appear in transactions that contain X = how often milk appears in transaction containing bread (probability of Y given that we already have X) "$P(Y|X)$"

$$c = \frac{\sigma(\{\text{Bread, Milk}\})}{\sigma(\{\text{Bread}\})} = 0.75$$

What is the Goal?

10

e.g. p% = 99%

- the support must guarantee the items that go together at least 99% of times
- the confidence must guarantee the item Y that is taken at least 99% of the times if it's already taken the item X

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ minsup threshold → the support must guarantee for example: "find me items that go together at least p% of times"
  - confidence ≥ minconf threshold → same thing for the confidence
- Brute-force approach
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the minsup and minconf thresholds
- Brute-force approach is computationally prohibitive!

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

In order to obtain association rules we must first focus on itemsets (frequent itemsets)

Mining Association Rules

the support is the same

the confidence is different

- {Bread, Jam} ⇒ {Milk} s=0.4 c=0.75
- {Milk, Jam} ⇒ {Bread} s=0.4 c=0.75
- {Bread} ⇒ {Milk, Jam} s=0.4 c=0.75
- {Jam} ⇒ {Bread, Milk} s=0.4 c=0.6
- {Milk} ⇒ {Bread, Jam} s=0.4 c=0.5

- They are all binary partitions of the itemset {Milk, Bread, Jam}
- Thus, rules originating from the same itemset have,
  - Same support but
  - Can have different confidence

We can decouple the support and confidence requirements!

First, find the itemsets with support ≥ minsup

Next, generate the rules with confidence ≥ minconf

What are Frequent Itemsets?

- **Frequent Itemset**
  - An itemset whose support is greater than or equal to the minsup threshold
- **Support**
  - Fraction of transactions that contain an itemset
  - s({Milk, Bread}) = 3/8
    s({Soda, Chips}) = 4/8
- **Support count (σ)**
  - Raw count of occurrence of an itemset
  - For example, σ({Milk, Bread}) = 3 σ({Soda, Chips}) = 4

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

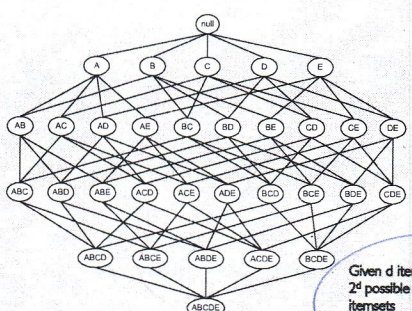
This means that Milk and Bread appear together 3 times over the 8 transactions

Mining Association Rules: a Two Step Approach

- Frequent Itemset Generation
  - Generate all itemsets whose support ≥ minsup
- Rule Generation
  - Generate high confidence rules from frequent itemset
  - Each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is computationally expensive

generation of all possible itemsets starting from: A, B, C, D, E

Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

that's NOT good

Brute Force

16

```
BRUTEFORCE (D, I, minsup):
1 F ← ∅ // set of frequent itemsets
2 foreach X ⊆ I do
3 |   sup(X) ← COMPUTESUPPORT (X, D)
4 |   if sup(X) ≥ minsup then
5 |   |   F ← F ∪ {(X, sup(X))}
6 return F

COMPUTESUPPORT (X, D):
1 sup(X) ← 0
2 foreach ⟨t, i(t)⟩ ∈ D do
3 |   if X ⊆ i(t) then
4 |   |   sup(X) ← sup(X) + 1
5 return sup(X)
```

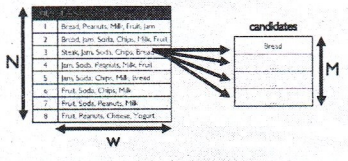
Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Brute-Force Generation of Frequent Itemsets

17

this really depends on the number of transactions and the number of items and the number of candidates to generate (the problem is that the number of candidates (M) is exponential)

- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database



- Match each transaction against every candidate
- Complexity ~ O(NMw) => Expensive since M = 2^d

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Frequent Itemset Generation Strategies

18

- **Reduce the number of candidates (M)**
  - Complete search has M=2^d
  - Use pruning techniques to reduce M
- **Reduce the number of transactions (N)**
  - Reduce size of N as the size of itemset increases
- **Reduce the number of comparisons (NM)**
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Reducing the Number of Candidates: (M) The Apriori Principle

19

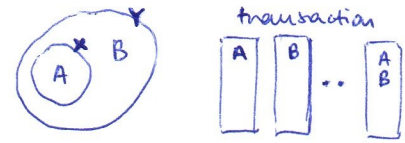
- Apriori principle
  - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

- Support of an itemset never exceeds the support of its subsets
- This is known as the anti-monotone property of support

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

This means that if, for example, we have apples (A) and bananas (B) then the number of times that we find A is for sure greater or equal to the number of times for which we find **A and B**.



(# A) ≥ (# {A, B})

= number of times we find .. in transactions

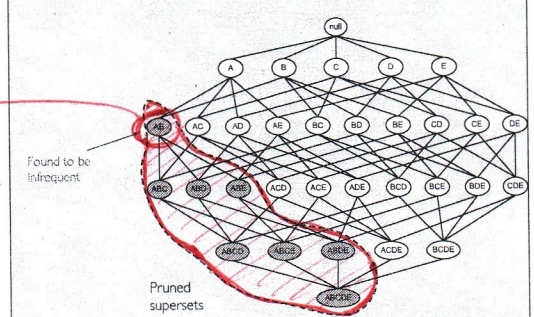
In this way, if already **A** is not frequent we know that **A and B** cannot be frequent!

This helps in the way that: if we find an itemset which is not frequent then EVERY itemset which contains the non-frequent itemset will be non frequent!

↳ good for pruning candidates

Illustrating Apriori Principle

20



Prof. Pierluca Lanzi — POLITECNICO DI MILANO

As soon as we find out that AB is not frequent we can prune and eliminate everything that has **AB**

1. We establish a minimum support, in this case: 4/8
2. We start with itemsets of **1** item and we evaluate their support
3. We consider for the second step only the itemsets of **2.** that has support ≥ 4/8. We create itemsets of **2** items and we evaluate their support
4. We consider from 3. only the itemsets with support ≥ 4/8. Then we procede to form itemsets of 3 elements (starting from the elements of **3.** (the "survived")

[and so on]

How does the Apriori principle work?

21

Minimum Support = 4/8

Item	Support
Bread	4/8
Peanuts	4/8
Milk	6/8
Fruit	6/8
Jam	5/8
Soda	5/8
Chips	4/8
Steak	1/8 ✗
Cheese	1/8 ✗
Yogurt	1/8 ✗

1-itemsets

2 Itemset	Support
Bread, Jam	4/8
Peanuts, Fruit	4/8
Milk, Fruit	5/8
Milk, Jam	4/8
Milk, Soda	5/8
Fruit, Soda	4/8
Jam, Soda	4/8
Soda, Chips	4/8

2-itemsets

3 Itemset	Support
Milk, Fruit, Soda	4/8

3-itemsets

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Example from the Textbook

22

- Given the following database and a min support of 3, generate all the frequent itemsets

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

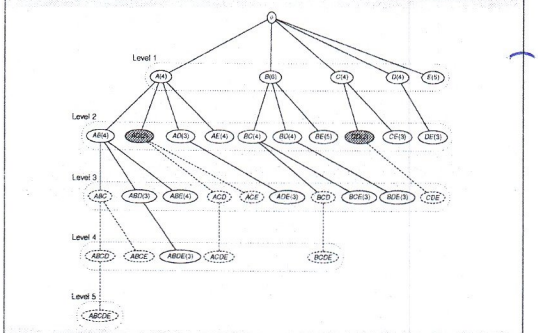
t	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

Transaction Database

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Example from the Textbook

23



Prof. Pierluca Lanzi — POLITECNICO DI MILANO

↑ It's a LEVEL-WISE search

The Apriori Algorithm

24

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the database
  - Eliminate candidates that are infrequent, leaving only those that are frequent

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

0. Set the minimum support
1. $k=1$
2. Generate all k-itemsets (if $k>1$ generate all k-itemsets containing only the preceding survived $(k-1)$-itemsets)
3. Compute the support for all the new itemsets
4. Prune the itemsets with support < minimum support
5. If there are no new frequent itemset then end, otherwise $k++$ and go to 2.

Apriori Algorithm

25

APRIORI (**D**, $\mathcal{I}$, *minsup*):

```
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6  |  COMPUTESUPPORT (C(k), D)
7  |  foreach leaf X ∈ C(k) do
8  |  |  if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9  |  |  else remove X from C(k)
10 |  C(k+1) ← EXTENDPREFIXTREE (C(k))
11 |  k ← k + 1
12 return F(k)
```

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Apriori Algorithm

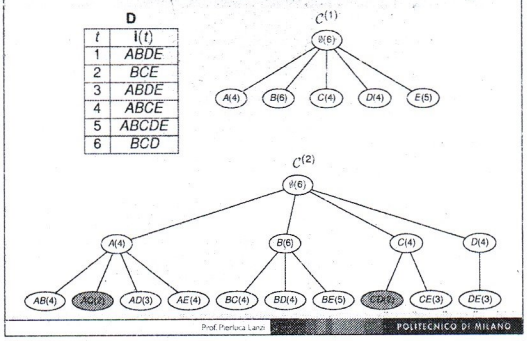

```
ComputeSupport (C^(k), D):
1 foreach ⟨t, i(t)⟩ ∈ D do
2   foreach k-subset X ⊆ i(t) do
3     if X ∈ C^(k) then sup(X) ← sup(X) + 1

ExtendPrefixTree (C^(k)):
1 foreach leaf X_a ∈ C^(k) do
2   foreach leaf X_b ∈ SIBLING(X_a), such that b > a do
3     X_ab ← X_a ∪ X_b
      // prune candidate if there are any infrequent
         subsets
4     if X_j ∈ C^(k), for all X_j ⊂ X_ab, such that |X_j| = |X_ab| − 1 then
5       Add X_ab as child of X_a with sup(X_ab) ← 0
6   if no extensions from X_a then
7     remove X_a, and all ancestors of X_a with no extensions, from C^(k)
8 return C^(k)
```


Apriori Algorithm Example (min sup=3): First two levels of the search

D

t	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD



Eclat

The Eclat Algorithm

- Leverages the tidsets directly for support computation.
- The support of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets.
- Given t(X) and t(Y) for any two frequent itemsets X and Y, then t(XY)=t(X) ∧ t(Y)
- And sup(XY) = |t(XY)|

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

t	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

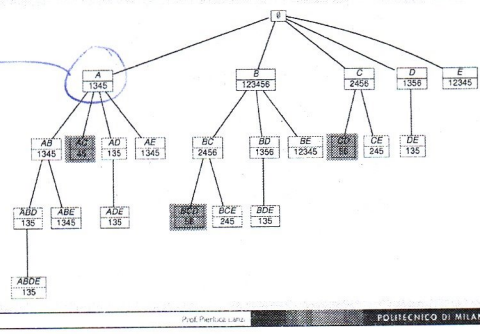
Transaction Database

t(X)

A	B	C	D	E
1	1	2	1	1
3	2	4	3	2
4	3	5	5	3
5	4	6	6	4
	5			5
	6			

Vertical Database

Example



It still uses the apriori principle but, because of the change in notation, we have that the procedure is so much faster

VERTICAL DATABASE for every item we list the number of the transactions that contain it

Now: the support of an item, for example A, is just the number of transactions that the item has, in this case, for A, 4.
If we want to find the support of AB we just do the intersection of the two lists. (with binary representation it's really fast)

The result is still the same. (since the principle is the same)

A
1345

The item A appears in the transactions: 1, 3, 4, 5 (4 trans.)

```
// Initial Call:  F ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, |t(i)| ≥ minsup}
ECLAT (P, minsup, F):
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

Frequent Patterns Mining Without Candidate Generation

Can we eliminate the candidate generation?

Apriori's Performance Bottlenecks

33

- The core of the Apriori algorithm
  - Use frequent (k–1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- It may need to generate huge candidate sets
  - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
  - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, needs to generate $2^{100} \sim 10^{30}$ candidates.
  - Multiple scans of database, it needs (n+1) scans, n is the length of the longest pattern

FP-TREE
We use a tree structure to summarize all the data without generating the candidates

Mining Frequent Patterns Without Candidate Generation

34

- 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
- Use 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

The FP-Growth Algorithm

35

- Leave the generate-and-test paradigm of Apriori
- Data sets are encoded using a compact structure, the FP-tree
- Frequent itemsets are extracted directly from the FP-tree
- Major Steps to mine FP-tree
  - Construct the frequent pattern tree
  - For each frequent item i compute the projected FP-tree
  - Recursively mine conditional FP-trees and grow frequent patterns obtained so far
  - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Part I

(p. 231)

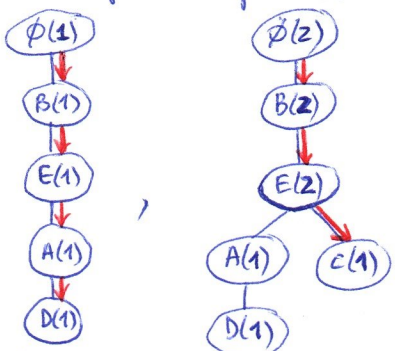
We take the table and we obtain an order based on how many times it appears (+ alphabetically):

{B(6), E(5), A(4), C(4), D(4)}

⟹ New order: BEACD

⟹ We obtain a new table with this order (new table of transactions)

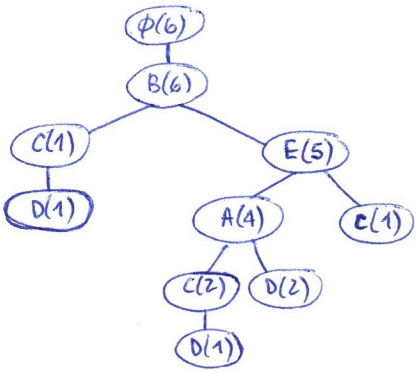
Then we start building the tree with the 1st transaction. The leaves will count how many times we pass through them:

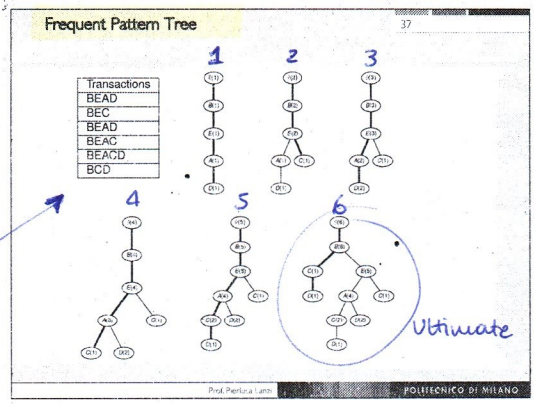


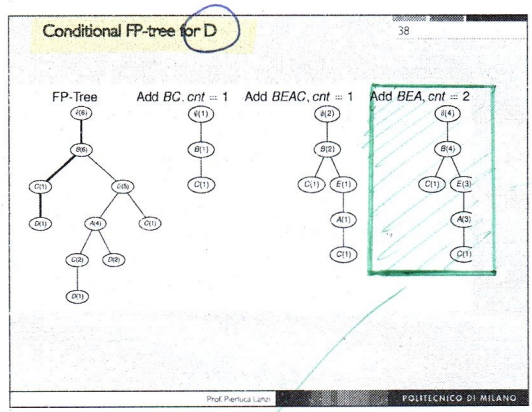
1st trans. {B, E, A, D}

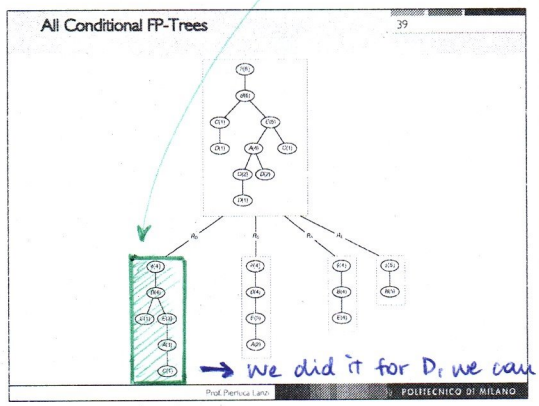
2nd trans. {B E C}

Ultimate:

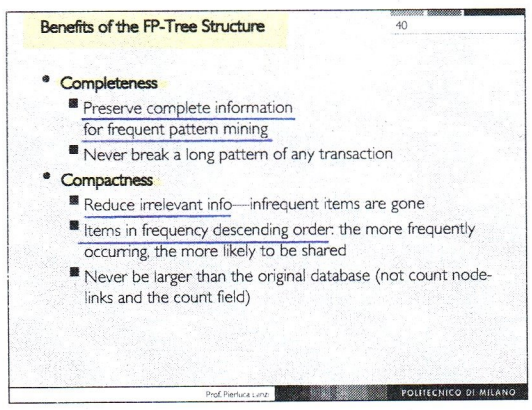


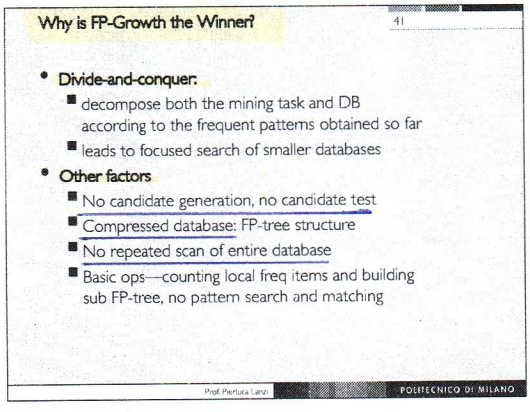






→ We did it for D, we can procede with all the others.





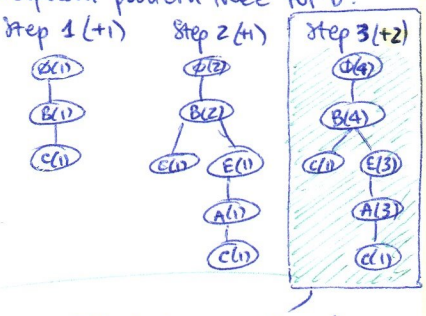
Part II

From here we focus on the CONDITIONAL FP-tree (for D). Which are the trees (subtrees) from the ultimate ending with D?

BCD count(D) = 1
BEACD count(D) = 1
BEAD count(D) = 2

And we now build the projected frequent pattern tree for D:

Step 1 (+1) Step 2 (+1) Step 3 (+2)



This is the conditional FP-tree for D. This can generate all the possible combinations of the items that generate frequent itemsets for D.

! So, given D we can have:

BEA, EA, BE, E, A, B

Because the most frequent (single) items given D are A, B, E

→ These are all the patterns that are prefixes of frequent itemsets that end with D.

→ Notice that we didn't consider C because the minimum support (we fixed as) 3.

From here we know to consider as frequent itemsets:

BEAD, EAD, BED, ED, AD, BD

Rule Generation

Rule Generation

43

- Given a frequent itemset L, find all non-empty subsets f ⊂ L such that f ⇒ L – f satisfies the minimum confidence requirement
- If {A,B,C,D} is a frequent itemset, candidate rules:
  ABC ⇒ D, ABD ⇒ C, ACD ⇒ B, BCD ⇒ A, A ⇒ BCD,
  B ⇒ ACD, C ⇒ ABD, D ⇒ ABC, AB ⇒ CD, AC ⇒ BD,
  AD ⇒ BC, BC ⇒ AD, BD ⇒ AC, CD ⇒ AB
- If |L| = k, then there are $2^k - 2$ candidate association rules (ignoring L → ∅ and ∅ → L)

we have all the possible combinations (which are a lot)

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

How to efficiently generate rules from frequent itemsets?

44

- Confidence does not have an anti-monotone property
- c(ABC⇒D) can be larger or smaller than c(AB ⇒ D)
- However, confidence of rules generated from the same itemset has an anti-monotone property
- L = {A,B,C,D}: c(ABC ⇒ D) >= c(AB ⇒ CD) >= c(A ⇒ BCD)
- Confidence is anti-monotone with respect to the number of items on the right-hand side of the rule

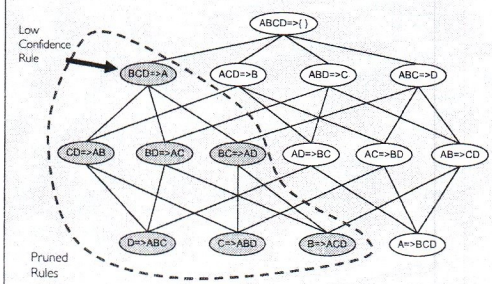
However, since it's more difficult to predict the presence of two product instead of the presence of one, we still have a rule: confidence (ABC ⇒ D) ≥ conf. (A ⇒ BCD)

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Rule Generation for Apriori Algorithm

45

analogue to the idea with supports



Notation : "head" ⇒ "tail"

When we start, we start with only one element on the tail and then, going on with levels, we add elements to the tail

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Association Rules Generation

46

```
AssociationRules (F, minconf):
1  foreach Z ∈ F, such that |Z| ≥ 2 do
2      A ← {X | X ⊂ Z, X ≠ ∅}
3      while A ≠ ∅ do
4          X ← maximal element in A
5          A ← A \ X // remove X from A
6          c ← sup(Z)/sup(X)
7          if c ≥ minconf then
8              print X ⟶ Y, sup(Z), c
9          else
10             A ← A \ {W | W ⊂ X}
               // remove all subsets of X from A
```

confidence = how many times the tail appears when we have the head (conditional probability)

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Example of Rule Generation (1)

- Suppose that we computed the following frequent itemsets and we have a minconf of 0.9

sup	itemsets
6	*B*
5	*E, BE*
4	*A, C, D, AB, AE, BC, BD, ABE*
3	*AD, CE, DE, ABD, ADE, BCE, BDE, ABDE*

- We start from the largeset itemset (ABDE) and consider all the possible subsets $\mathcal{A}$ = {ABDE(3), ABD(3), ABE(4), ADE(3), BDE(3), AB(4), AD(3), AE(4), BD(4), BE(5), DE(3), A(4), B(6), D(4), E(5)}

Example of Rule Generation (2)

- The first subset is X = ABD, and the confidence of ABD ⇒ E is 3/3 = 1.0, so we output the rule.
- The next subset is X = ABE, but the corresponding rule ABE ⇒ D is not strong since c(ABE ⇒ D) = 3/4 = 0.75
- We can thus remove from $\mathcal{A}$ all subsets of ABE !!!!
- The updated set of antecedents is therefore $\mathcal{A}$ = {ADE(3), BDE(3), AD(3), BD(4), DE(3), D(4)}
- When the algorithm ends, it will output the rules, ABD ⇒ E conf=1.0; ADE ⇒ B conf=1.0; BDE ⇒ A, conf=1.0; AD ⇒ BE conf=1.0; DE ⇒ AB conf=1.0

Rule Assessment Measures

How to Set an Appropriate Minsup?

- If minsup is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)
- If minsup is set too low, it is computationally expensive and the number of itemsets is very large
- A single minimum support threshold may not be effective

If we set min sup = 1 then we're looking for itemsets that are always there, it's not a great discover

If we set min sup = 0.01 then items are very easily frequent → we obtain a huge amount of frequent itemsets. How can we select among them?

Lift

- Lift is the ratio of the observed joint probability of X and Y to the expected joint probability if they were statistically independent.

lift(X ⇒ Y) = sup(X, Y)/sup(X)sup(Y) = conf(X ⇒ Y)/sup(Y)

this is not a proper union/inters., it's an "AND"

$\sim \frac{IP(Y|X)}{IP(Y)}$

- Lift is a measure of the deviation from stochastic independence (if it is 1 then X and Y are independent)
- Lift also measure the surprise of the rule. A lift close to 1 means that the support of a rule is expected considering the supports of its components.
- We typically look for values of lift that are much larger (i.e., above expectation) or much smaller (i.e., below expectation) than one.

Summarizing Itemsets

we want to find something that summarizes informations. for example: if we say that "ABDE" is a frequent itemset we also say that {A, B, D, E} and all their combinations are frequent itemsets.

Maximal Frequent Itemsets

51

- A frequent itemset X is called maximal if it has no frequent supersets
- The set of all maximal frequent itemsets, given as

  $M = \{X \mid X \in F \text{ and } !\exists Y \supset X, \text{ such that } Y \in F\}$

- M is a condensed representation of the set of all frequent itemset F, because we can determine whether any itemset is frequent or not using M
- If there is a maximal itemset Z such that $X \subseteq Z$, then X must be frequent, otherwise Z cannot be frequent
- However, M alone cannot be used to determine sup(X), we can only use to have a lower-bound, that is, $sup(X) >= sup(Z)$ if $X \subseteq Z \in M$.

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Maximal frequent itemsets are interesting because all their subsets are frequent too

Closed Frequent Itemsets

55

- An itemset X is closed if all supersets of X have strictly less support, that is,

  $sup(X) > sup(Y)$, for all $Y \supset X$

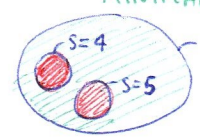
- The set of all closed frequent itemsets C is a condensed representation, as we can determine whether an itemset X is frequent, as well as the exact support of X using C alone

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Once we have a maximal frequent itemset we're interested in knowing when/where the support changes.
We know that all the supersets have smaller supports than their subsets.
If a subset is such that ALL its supersets have STRICTLY smaller supports then it's called CLOSED FREQ. ITEMSET.

→ If we have a closed frequent itemset X then every itemset containing X has a STRICTLY smaller support than X.

CLOSED FREQ. ITEMSETS
MINIMAL GENERATOR



Minimal Generators

56

- A frequent itemset X is a minimal generator if it has no subsets with the same support

  $G = \{X \mid X \in F \text{ and } !\exists Y \subset X, \text{ such that } sup(X) = sup(Y)\}$

- Thus, all subsets of X have strictly higher support, that is,

  $sup(X) < sup(Y)$

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

minimal generator = all the subsets have a strictly larger support than the minimal

57

we start from:

Transaction database

Tid	Itemset
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

and we obtain:

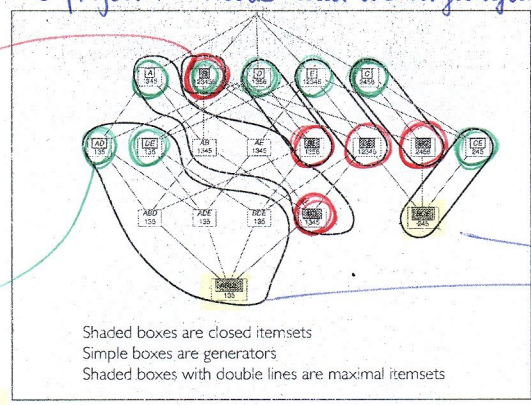
Frequent itemsets (*minsup* = 3)

sup	Itemsets
6	*B*
5	*E, BE*
4	*A, C, D, AB, AE, BC, BD, ABE*
3	*AD, CE, DE, ABD, ADE, BCE, BDE, ABDE*

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

We project the results and we highlight:

- maximal itemsets
- generators
- closed itemsets



For example: B is a closed itemset because every itemset that contains B has a lower support

For example: AD is a generator because it has no subsets with the same support (S(A)=4, S(D)=4)

these are maximal because there's nothing below. If there would have been then they wouldn't be maximal (These 2 itemsets somehow already summarize the whole tree)

To check that an itemset is a MAXIMAL ITEMSET we look if it is at the end of the tree, to check if it's a GENERATOR we have to look at the upper levels of the itemsets (since we look at the subsets), to check if it's a CLOSED ITEMSET we look at the lower level (since we look at the bigger itemsets containing the one we're analyz.)

Searching for Small Communities (Trawling)

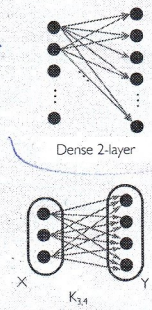
We can translate it as an ASSOCIATION MINING problem

Searching for small communities in the Web graph (Trawling)

50

- **Trawling**
  - What is the signature of a community in a Web graph?
  - The underlying intuition, that small communities involve many people talking about the same things
  - Use this to define "topics": what the same people on the left talk about on the right?
- **More formally**
  - Enumerate complete bipartite subgraphs $K_{s,t}$
  - $K_{s,t}$ has s nodes on the "left" and t nodes on the "right"
  - The left nodes link to the same node of on the right, forming a fully connected bipartite graph

$K_{s,t}$



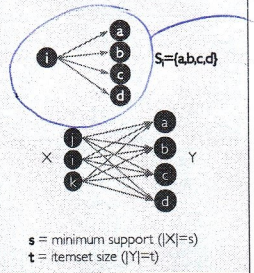
Prof. Pierluca Lanzi — POLITECNICO DI MILANO

community of **s** talking about **t** topics

Mining Bipartite $K_{s,t}$ using Frequent Itemsets

61

- Searching for such complete bipartite graphs can be viewed as a frequent itemset mining problem
- View each node i as a set S_i of nodes i points to
- $K_{s,t}$ = a set Y of size t that occurs in s sets S_i
- Looking for $K_{s,t}$ is equivalent to setting the frequency threshold to s and look at layer t (i.e., all frequent sets of size t)



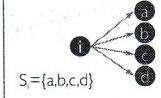
Prof. Pierluca Lanzi — POLITECNICO DI MILANO

For the node **i** we build a set of all the nodes to which **i** is connected: {a, b, c, d}. *

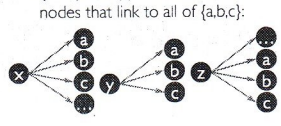
* What is in this scenario $K_{s,t}$? a set of "t" that occurs in "s" sets (basically: a set of "t" that has support "s")

→ When we look for $K_{s,t}$ we look for all the itemsets of size **t** with a minimum support of **s**

View each node **i** as a set S_i of nodes **i** points to

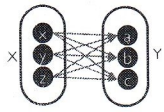


Suppose we find a frequent itemset Y={a,b,c} of supp s; then, there are s nodes that link to all of {a,b,c}:



Find frequent itemsets with support s and itemset size t is equivalent to find $K_{s,t}$

$K_{s,t}$ = a set Y of size t that occurs in s sets S_i



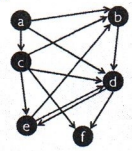
1. From the graph (1) generate the database (2)
2. From the database (2) generate the frequent itemsets (3)
3. From the frequent itemsets (3) generate the subgraphs (4)

{b,d} supports 3 means that are at least 3 elements that contains {b,d} (in fact: a,c,e)

Example

63

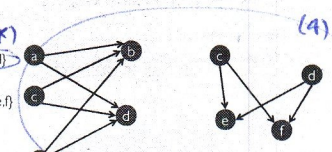
(1)



- Support threshold s=2
  - {b,d}: support 3
  - {e,f}: support 2
- And we just found 2 bipartite subgraphs:

this means that basically we're looking for communities of at least 2 people discussing stuff

(2)
- Itemsets (*)
  - a = {b,c,d}
  - b = {d}
  - c = {b,d,e,f}
  - d = {e,f}
  - e = {b,d}
  - f = {}

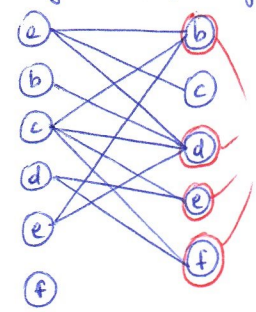


(4)

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

This because a is connected to: b, c, d

Starting from (*) we get:



(3) frequent itemsets ("topics")

We can divide the community

Mining Frequent Sequences

This is based on the idea that an itemsets it's not a set but a sequence (the order inside the transaction is important)

Sequence Mining: Terminology

65

- A sequence is an ordered list of symbols $\mathbf{s} = s_1, s_2, \ldots s_k$ where s_i or $\mathbf{s}[i]$ denote the symbol at position i
- Notation $\mathbf{s}[i;j]$ denotes $s_{i+1}, s_{i+2}, \ldots s_j$ (continuous subsequence)
- Let $\mathbf{s} = s_1, s_2, \ldots, s_n$ and $\mathbf{r} = r_1, r_2, \ldots, r_m$ we say that $\mathbf{r}$ is a subsequence of s denoted $r \subseteq s$ if there is a function ϕ from $[1,m]$ to $[1,n]$ such that $r[i] = s[\phi(i)]$ and for any position i, j in r,

$$i<j \Rightarrow \phi(i)<\phi(j)$$

- $\mathbf{r}$ is a consecutive subsequence when

$$r_1, r_2, \ldots, r_m = s_j, s_{j+1}, \ldots, s_{j+m-1}$$

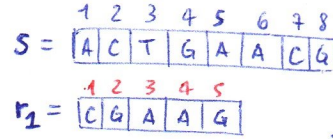
Prof. Pierluca Lanzi — POLITECNICO DI MILANO

Example

66

- Given $\mathbf{s}$ = ACTGAACG
- $\mathbf{r}_1$ = CGAAG is a subsequence of $\mathbf{s}$
- $\mathbf{r}_2$ = CTGA is a consecutive subsequence of $\mathbf{s}$
- $\mathbf{r}_3$ = ACT is a prefix of $\mathbf{s}$
- $\mathbf{r}_4$ = GAACG is a suffix of $\mathbf{s}$

Prof. Pierluca Lanzi — POLITECNICO DI MILANO



In this case ϕ is:

$\phi(1) = 2$ $\quad \phi(4) = 6$

$\phi(2) = 4$ $\quad \phi(5) = 8$

$\phi(3) = 5$

$\Rightarrow \exists \phi \Rightarrow r_1$ is a subsequence

Frequent Sequences

67

- Given a database D containing N sequences, the support of a sequence $\mathbf{r}$ in D is defined as the total number of sequences in D that contains r

$$sup(\mathbf{r}) = |\{\mathbf{s}_i \in \mathbf{D} | \mathbf{r} \subseteq s_i\}|$$

- The relative support of r is the percentage of sequences that contain r,

$$rsup(\mathbf{r}) = sup(\mathbf{r})/N$$

- Given a minsup threshold we that r is frequent if sup(r)>=minsup
- Note that, in sequence mining, we need to consider all possible permutations of the items not just the combinations

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

this check how many times we see a sequence (subsequence) in the database

Example

- Given the following sequence database

Id	Sequence
s_1	CAGAAGT
s_2	TGACAG
s_3	GAAGT

- With a minsup of 3, the set of frequent subsequences is
  - (A(3), G(3), T(3)
  - AA(3), AG(3), GA(3), GG(3)
  - AAG(3), GAA(3), GAG(3)
  - GAAG(3)

Support? 3 : s_1 : CAGAAGT, s_2 : TGACAG, s_3 : GAAGT

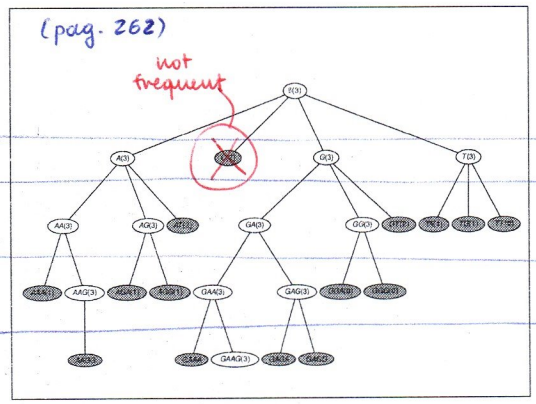
for example: what is the support of A? 3. Even if A appears more than once in every sequence we don't care (we don't consider repetitions)

for example: CAG has a support of 2 ⟹ it's not frequent (minsup = 3)

Level-Wise Sequence Mining: GSP Algorithm

- Searches the sequence prefix tree using a level-wise (breath-first search).
- Given the set of frequent sequences at level k, the algorithm generates the candidate for level k+1 and compute the support of each candidate and prune the not frequent ones.
- For each sequence s_i in D, we check if a candidate **r** is a subsequence of **s**, if it is the support of r is incremented. Once the frequent sequences at level k are computed the k+1 candidates are generated.
- For each leaf $\mathbf{r}_a$, the sequence is extended with the last symbol of any other leaf $\mathbf{r}_b$ that shares the same prefix (it has the same parent) so $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$. If $\mathbf{r}_{ab}$ is infrequent, we prune it.

(pag. 262)



1st level: only 1-sequences

2nd level: 2-sequences

3rd

4th

Notice: this time the order (in the sequence) matters, so from A we'll start AA, AG, AT but for AG we won't connect to G! From G we'll start GA! (In apriori we wouldn't do it!)

⟹ same procedure as with apriori but the number of candidates is even larger (this is an issue)

The GSP Algorithm

GSP (D, Σ, *minsup*):

```
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single symbols
3  foreach s ∈ Σ do Add s as child of ∅ in C(1) with sup(s) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      ComputeSupport (C(k), D)
7      foreach leaf s ∈ C(k) do
8          if sup(r) ≥ minsup then F ← F ∪ {(r, sup(r))}
9          else remove s from C(k)
10     C(k+1) ← ExtendPrefixTree (C(k))
11     k ← k + 1
12 return F(k)
```


The GSP Algorithm

ComputeSupport ($C^{(k)}$, D):

```
1 foreach s_i ∈ D do
2     foreach r ∈ C(k) do
3         if r ⊆ s_i then sup(r) ← sup(r) + 1
```

ExtendPrefixTree ($C^{(k)}$):

```
1 foreach leaf r_a ∈ C(k) do
2     foreach leaf r_b ∈ Children(Parent(r_a)) do
3         r_ab ← r_a + r_b[k] // extend r_a with last item of r_b
          // prune if there are any infrequent
             subsequences
4         if r_c ∈ C(k), for all r_c ⊂ r_ab, such that |r_c| = |r_ab| − 1 then
5             Add r_ab as child of r_a with sup(r_ab) ← 0
6     if no extensions from r_a then
7         remove r_a, and all ancestors of r_a with no extensions, from C(k)
8 return C(k)
```

*:

- frequent itemset: we go to Esselunga and buy stuff, what do we buy together?
- frequent sequences: we have a DNA sequence and we look for frequent patterns inside of sequence (it's different from before because a sequence is not just an itemset)
- sequential pattern mining: we look for sequences of itemsets and we're going to look at relationships among items

Sequential Pattern Mining

association among transactions

we had FREQUENT ITEMSETS where we had transactions of itemsets and we were looking for frequent itemsets, then we had FREQUENT SEQUENCES where every itemset is a sequence, now we have SEQUENTIAL PATTERN MINING where we don't consider just a sequence but sequencies. *

Sequential Pattern Mining

74

- Association rules do not consider the order of transactions, however, in many applications ordering is significant
- In market basket analysis, it is interesting to know whether people buy some items in sequence
- Example
  - Buying bed first and then bed sheets later
  - Navigational patterns of users in a Web site from sequences of page visits of users

Examples of Sequence

75

- **Web sequence**
  - < {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >
- **Sequence of initiating events causing the accident at 3-mile Island:**
  - <{clogged resin} {outlet valve closure} {loss of feedwater} {condenser polisher outlet valve shut} {booster pumps trip} {main waterpump trips} {main turbine trips} {reactor pressure increases}>
- **Sequence of books checked out at a library:**
  - <{Fellowship of the Ring} {The Two Towers} {Return of the King}>

Basic Concepts

76

- Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items.
- A sequence is an ordered list of itemsets.
- We denote a sequence s by $\langle a_1, a_2, \ldots, a_r \rangle$, where a_j is an itemset, also called an element of s.
- An element (or an itemset) of a sequence is denoted by $\{x_1, x_2, \ldots, x_k\}$, where $x_j \in I$ is an item.
- We assume without loss of generality that items in an element of a sequence are in lexicographic order

For example:
sequence of 3 elements:

bread, milk	$a_1 = \{x_1, x_2\}$
water	a_2
cookies	a_3

When we perform sequential pattern mining we look for relations between elements of one itemsets and elements of another one

Sequential Pattern Mining

77

- Sequence databases consist of sequences of ordered elements or events (with or without time)
- Sequential Pattern Mining is the mining of frequently occurring ordered events or subsequences as patterns
- Sequences consists of nominal (categorical) data. When sequence are based on numerical data, then we apply time series analysis

Basic concepts

78

- The size of a sequence is the number of elements (or itemsets) in the sequence
- The length of a sequence is the number of items in the sequence (a sequence of length k is called k-sequence)
- Given two sequences, $s_1 = \langle a_1, a_2, \ldots, a_r \rangle$ and $s_2 = \langle b_1, b2, \ldots, b_v \rangle$
- s_1 is a subsequence of s_2 if there exist integers

$$1 \leq j_1 < j_2 < \cdots < j_r \leq v$$

such that

$$a_1 \subseteq b_{j_1} \wedge a_2 \subseteq b_{j_2} \wedge \cdots \wedge a_r \subseteq b_{j_v}$$

a_j, b_j are itemsets!

Example

79

- Let I = {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- The sequence ⟨{3}{4, 5}{8}⟩ is contained in (or is a subsequence of) ⟨{6} {3, 7}{9}{4, 5, 8}{3, 8}⟩
- In fact, {3} ⊆ {3, 7}, {4, 5} ⊆ {4, 5, 8}, and {8} ⊆ {3, 8}
- However, ⟨{3}{8}⟩ is not contained in ⟨{3, 8}⟩ or vice versa
- The size of the sequence ⟨{3}{4, 5}{8}⟩ is 3, and the length of the sequence is 4.

size = # itemsets, length = # items

This means: we once bought 3 and in the transaction after we bought 8 ⇒ ⟨{3}, {8}⟩ (3 happened at some point and later happened 8)
On the other hand: ⟨{3,8}⟩ means that 3 and 8 happened together, which means that there is no sequential aspect of this.

Goal of Sequential Pattern Mining

80

- The input is a set S of input data sequences (or sequence database)
- The problem of mining sequential patterns is to find all the sequences that have a user-specified minimum support
- Each such sequence is called a frequent sequence, or a sequential pattern
- The support for a sequence is the fraction of total data sequences in S that contains this sequence

Terminology

81

- **Itemset**
  - Non-empty set of items
  - Each itemset is mapped to an integer
- **Sequence**
  - Ordered list of itemsets
- **Support for a Sequence**
  - Fraction of total customers that support a sequence.
- **Maximal Sequence**
  - A sequence that is not contained in any other sequence
- **Large Sequence**
  - Sequence that meets minisup

Frequent Sequences

82

- Given a database D $\{s_1, \ldots, s_N\}$ of N sequences and a sequence r, we define the support count of r as

$$\text{sup}(r) = |\{s_i \mid r \text{ is a subsequence of } s_i\}|$$

- And the support (or relative support),

$$\text{rsup}(r) = \text{sup}(r)/N$$

Example 1.

Customer ID	Transaction Time	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10,20
2	June 15 '93	30
2	June 20 '93	40,60,70
3	June 25 '93	30,50,70
4	June 25 '93	30
4	June 30 '93	40,70
4	July 25 '93	90
5	June 12 '93	90

Customer ID	Customer Sequence
1	< (30) (90) >
2	< (10 20) (30) (40 60 70) >
3	< (30 , 50, 70) >
4	< (30) (40 70) (90) >
5	< (90) >

Maximal seq with support > 40%
< (30) (90) >
< (30) (40 70) >

Note: Use Minisup of 40%, no less than two customers must support the sequence
< (10 20) (30) > Does not have enough support (Only by Customer #2)
< (30) >, < (70) >, < (30) (40) > ... are not maximal.

Example 2.

- Given the following sequence database and a minsup of 2

SID	Sequence
10	<(a)(abc)(ac)(d)(cf)>
20	<(ad)(c)(bc)(ae)>
30	<(ef)(ab)(df)(c)(b)>
40	<(e)(g)(af)(c)(b)(c)>

- <(a)(bc)(d)(c)> is a subsequence of <(a)(abc)(ac)(d)(cf)>
- <(ab)(c)> is a frequent sequence or a sequential pattern

Example 3.

- Given the sequence database

SID	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(a)(b)(f)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

- Given a minsup of 2, <(bd)cb> is a sequential pattern (or frequent sequence)

actually:
<(bd)(c)(b)>

Apriori principle still applies

If a sequence S is not frequent, then every super-sequence of S is not frequent

In the previous example,

How do we proceed?
level-wise search!

Mining Sequential Patterns (Outline of the basic principle)

- Level-by-level do
  - Generate candidate sequences
  - Scan database to collect support counts
  - Use Apriori property to prune candidates
- Only generate candidates satisfying Apriori property
- Limitations
  - It can generate a massive amount of candidates (1000 frequent 1-sequences can generate around 1.5 million candidates)
  - It requires many scans of the database
- Accordingly, as for association rules, there are several prefix-based extensions (e.g. PrefixSpan) that deal with all these limitations

Example

88

Seq. ID	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(abc)(b)(df)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

1st scan - 8 candidates
6 sequential patterns (length 1)

<a> <b> <c> <d> <e> <f> ~~<g>~~ ~~<h>~~

2nd scan - 51 candidates
19 sequential patterns (length 2)
10 candidates were not in the database

<aa> <ab> .. <af> <ba> <bb> .. <ff>

3rd scan - 46 candidates
19 sequential patterns (length 3)
20 candidates were not in the database

<abb> <aab> <aba> ... <bab> ...

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

all possible letters appearing in the database

infrequent

this becomes really complex (here the a priori becomes more complex)

since we consider all the combinations also considering the order we may end with a lot of candidates which are not even in the database (⇒ we waste a lot of time)

> That's why we prefer the **FP-tree** approach: there is no candidates generation

Association Rules for Classification

Association rules means: $X \Rightarrow Y$ (if we buy X then we buy Y). We can translate it (use it) for classification if we consider X as a set of features and Y a label

Mining Association Rules for Classification (the CBA algorithm)

90

- Association rule mining assumes that the data consist of a set of transactions. Thus, the typical tabular representation of data used in classification must be mapped into such a format.
- Association rule mining is then applied to the new dataset and the search is focused on association rules in which the tail identifies a class label

$X \Rightarrow c_i$ (where c_i is a class label)

- The association rules are pruned using the pessimistic error-based method used in C4.5
- Finally, rules are sorted to build the final classifier.

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

The Weather Dataset

91

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

CLASS (LABELS)

one row is one transaction. We translate the dataset in one-hot-encoding approach ⇒ we obtain a binary dataset, which was one representation of the dataset for association rule

this transform. leads to this:

CBA Model

92

```
outlook=overcast ==> play=yes
humidity=normal windy=FALSE ==> play=yes
outlook=rainy windy=FALSE ==> play=yes
outlook=sunny humidity=high ==> play=no
outlook=rainy windy=TRUE ==> play=no

(default class is the majority class)
```

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

This is better than a decision tree or a logistic regression (in terms of READABILITY)