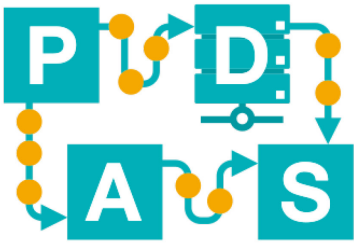
Sequence Mining

Lecture 13

IDS-L13



Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Outline of Today's Lecture

- Short recap: Apriori algorithm
- Event Data
- Mining sequential patterns
- AprioriAll Algorithm
- Extensions

Material

Mining Sequential Patterns

Rakesh Agrawal Ramakrishnan Srikant*

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

We are given a large database of customer transactions, where each transaction consists of customer-id, transaction time, and the items bought in the transaction. We introduce the problem of mining sequential patterns over such databases. We present three algorithms to solve this problem, and empirically evaluate their performance using synthetic data. Two of the proposed algorithms, AprioriSome and AprioriAll, have comparable performance, albeit AprioriSome performs a little better when the minimum number of customers that must support a sequential pattern is low. Scale-up experiments show that both AprioriSome and AprioriAll scale linearly with the number of customer transactions. They also have excellent scale-up properties with respect to the number of transactions per customer and the number of items in a transaction.

1 Introduction

Database mining is motivated by the decision support problem faced by most large retail organizations. Progress in bar-code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred to as the *basket* data. A record in such data typically consists of the transaction date and the items bought in the transaction. Very often, data records also contain customer-id, particularly when the purchase has been made using a credit card or a frequent-buyer card. Catalog companies also collect such data using the orders they receive.

We introduce the problem of mining *sequential* patterns over this data. An example of such a pattern is that customers typically rent "Star Wars", then "Empire Strikes Back", and then "Return of the Jedi". Note that these rentals need not be consecutive. Customers who rent some other videos in between also support this sequential pattern. Elements of a sequential pattern need not be simple items. "Fitted Sheet and flat sheet and pillow cases", followed by "comforter", followed by "drapes and ruffles" is an example of a sequential pattern in which the elements are sets of items.

Problem Statement We are given a database $\mathcal{D}$ of customer transactions. Each transaction consists of the following fields: customer-id, transaction-time, and the items purchased in the transaction. No customer has more than one transaction with the same transaction-time. We do not consider quantities of items bought in a transaction: each item is a binary variable representing whether an item was bought or not.

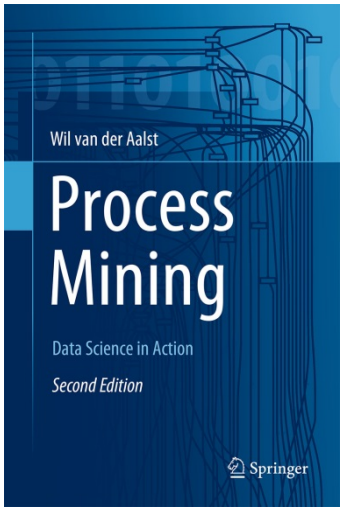
An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets. Without loss of generality, we assume that the set of items is mapped to a set of contiguous integers. We denote an itemset i by $(i_1 i_2 \ldots i_m)$, where i_j is an item. We denote a sequence s by $\langle s_1 s_2 \ldots s_n \rangle$, where s_j is an itemset.

A sequence $\langle a_1 a_2 \ldots a_n \rangle$ *is contained in* another sequence $\langle b_1 b_2 \ldots b_m \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. For example, the sequence $\langle (3)\ (4\ 5)\ (8) \rangle$ is contained in $\langle (7)\ (3\ 8)\ (9)\ (4\ 5\ 6)\ (8) \rangle$, since $(3) \subseteq (3\ 8)$, $(4\ 5) \subseteq (4\ 5\ 6)$ and $(8) \subseteq (8)$. However, the sequence $\langle (3)\ (5) \rangle$ is not contained in $\langle (3\ 5) \rangle$ (and vice versa). The former represents items 3 and 5 being bought one after the other, while the latter represents items 3 and 5 being bought together. In a set of sequences, a sequence s is *maximal* if s is not contained in any other sequence.

All the transactions of a customer can together be viewed as a sequence, where each transaction corresponds to a set of items, and the list of

*Also Department of Computer Science, University of Wisconsin, Madison.

Rakesh Agrawal, Ramakrishnan Srikant: Mining Sequential Patterns. ICDE 1995: 3-14



"W. van der Aalst. Process Mining: Data Science in Action. Springer-Verlag, Berlin, 2016" (http://springer.com/9783662498507)

The book "Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han, Micheline Kamber, and Jian Pei. Morgan Kaufmann, June 2011"is rather brief about sequence mining, but many of the general principles are valid also for sequence mining.

Chair of Process and Data Science

Short recap: Apriori algorithm

Apriori algorithm: Basic ingredients

- Let $D \in \mathfrak{B}\left(\mathcal{P}(\mathcal{I})\right)$ be a dataset and min_sup a chosen threshold.
- If $A \subseteq B$ and $support(A) < min\_sup$, then $support(B) < min\_sup$.
- Leveling: $L_k = \{A \subseteq \mathcal{I} | support(A) \geq min\_sup \wedge |A| = k\}$.

Apriori algorithm: Basic ingredients

- L_k are all frequent itemsets of length k.
- **Construction of candidates**: Any $A = \{a_1, a_2, \dots, a_{k-1}, a_k\} \in L_k$ can be obtained by joining $A' = \{a_1, a_2, \dots a_{k-2}, a_{k-1}\} \in L_{k-1}$ and $A'' = \{a_1, a_2, \dots a_{k-2}, a_k\} \in L_{k-1}$.
- Let C_k be the set of all **candidates** created from L_{k-1} in this manner.
- **Prune** the set C_k by removing itemsets that have infrequent subsets.

Algorithm

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)     Ck = apriori_gen(Lk−1);
(4)     for each transaction t ∈ D { // scan D for counts
(5)       Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)       for each candidate c ∈ Ct
(7)         c.count++;
(8)     }
(9)     Lk = {c ∈ Ck|c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)     for each itemset l2 ∈ Lk−1
(3)       if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
            ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)         c = l1 ⋈ l2; // join step: generate candidates
(5)         if has_infrequent_subset(c, Lk−1) then
(6)           delete c; // prune step: remove unfruitful candidate
(7)         else add c to Ck;
(8)       }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
        Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)     if s ∉ Lk−1 then
(3)       return TRUE;
(4)   return FALSE;
```

creating candidates C_k based on L_{k-1}

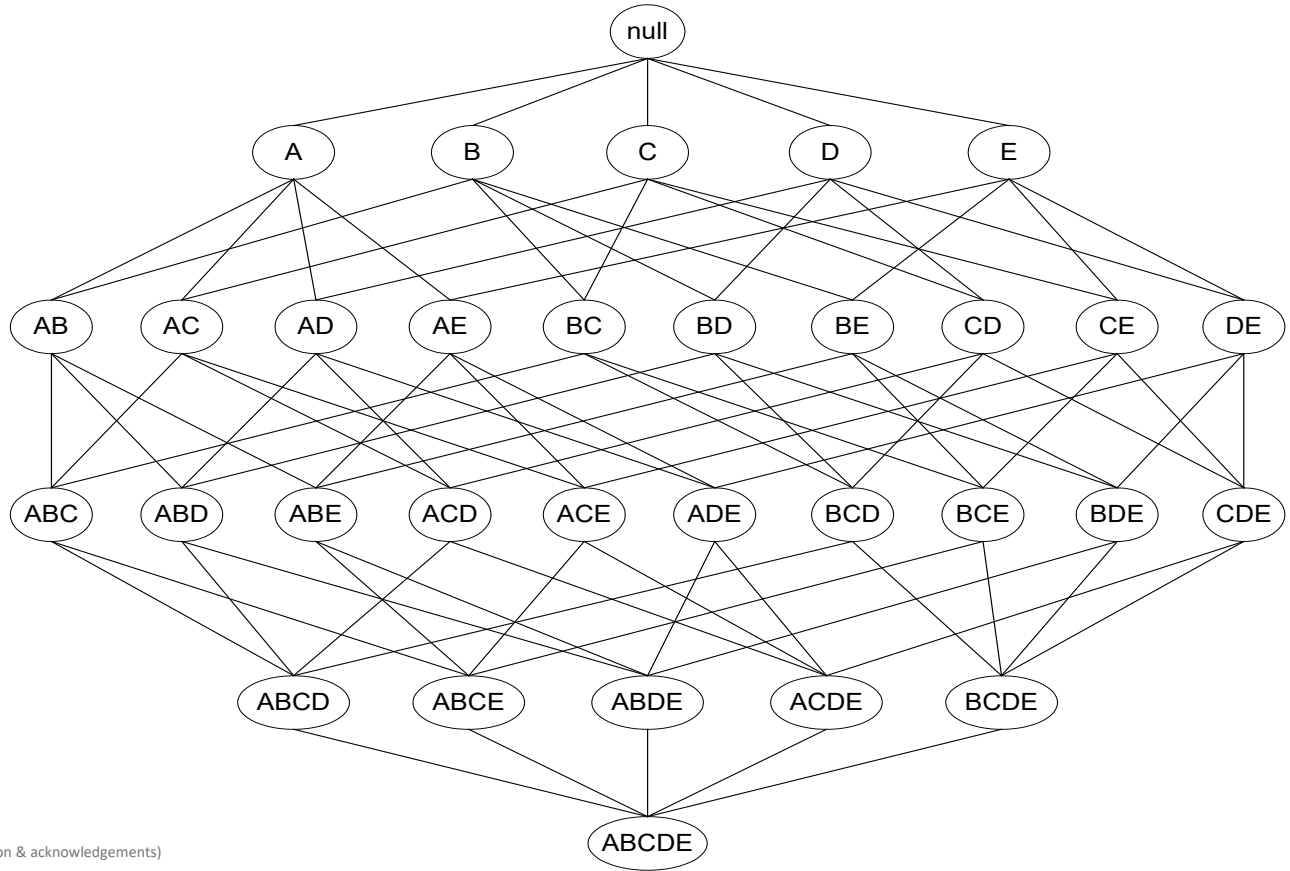
testing candidates to create L_k based on C_k

create candidates from L_{k-1} (avoiding duplicates)

prune set of candidates by checking whether subsets were frequent before

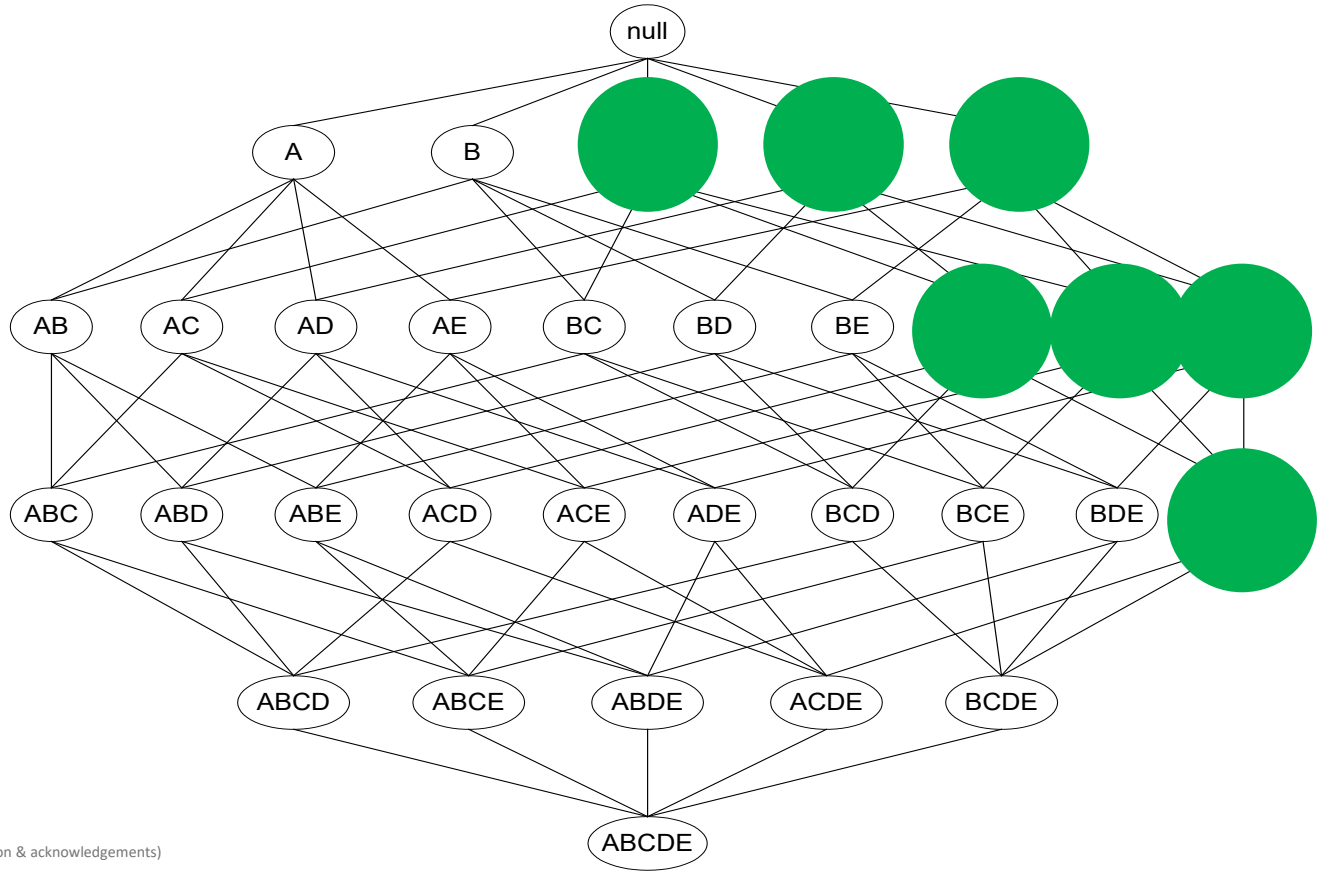
Chair of Process and Data Science

Consider a data set with 5 items



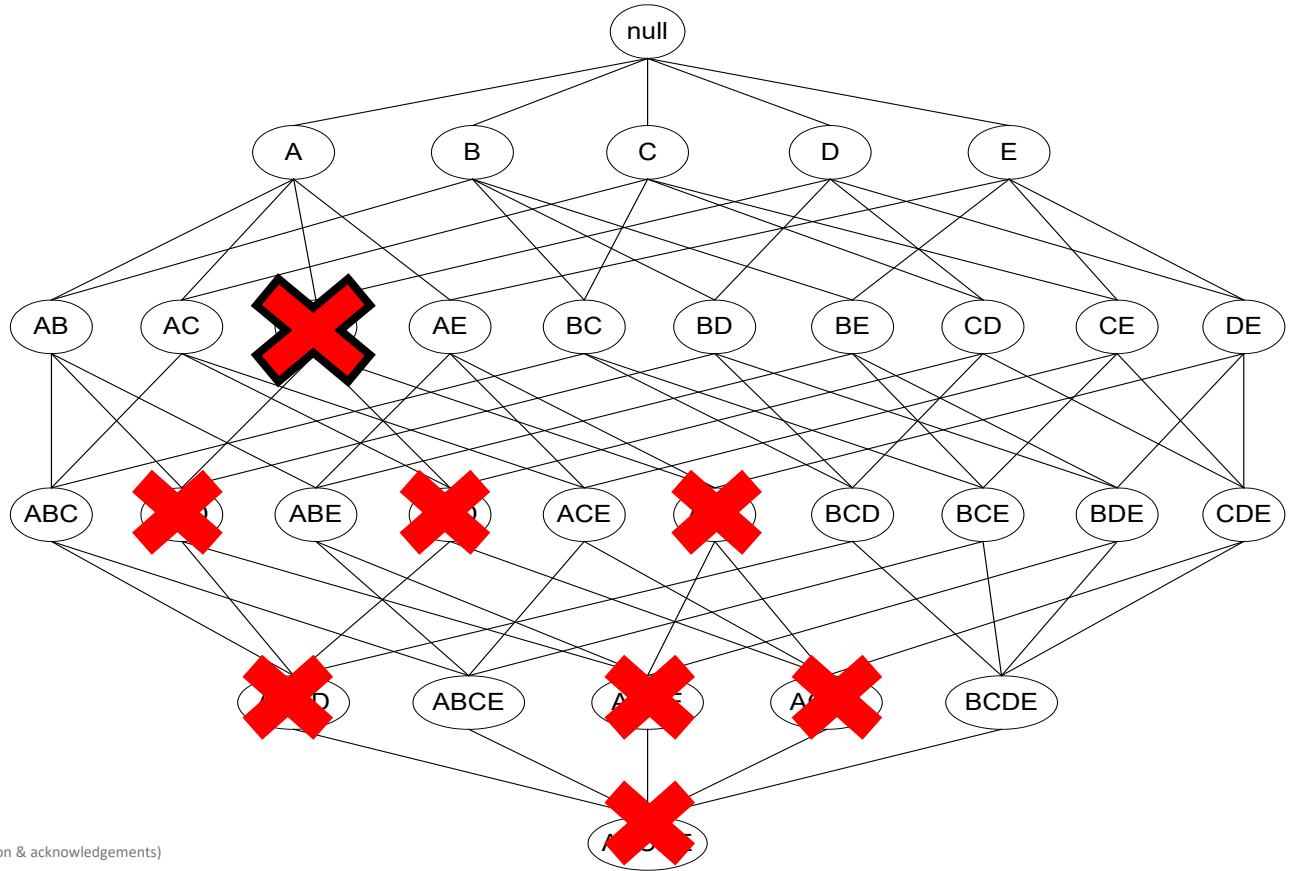
Chair of Process and Data Science

Consider a data set with 5 items



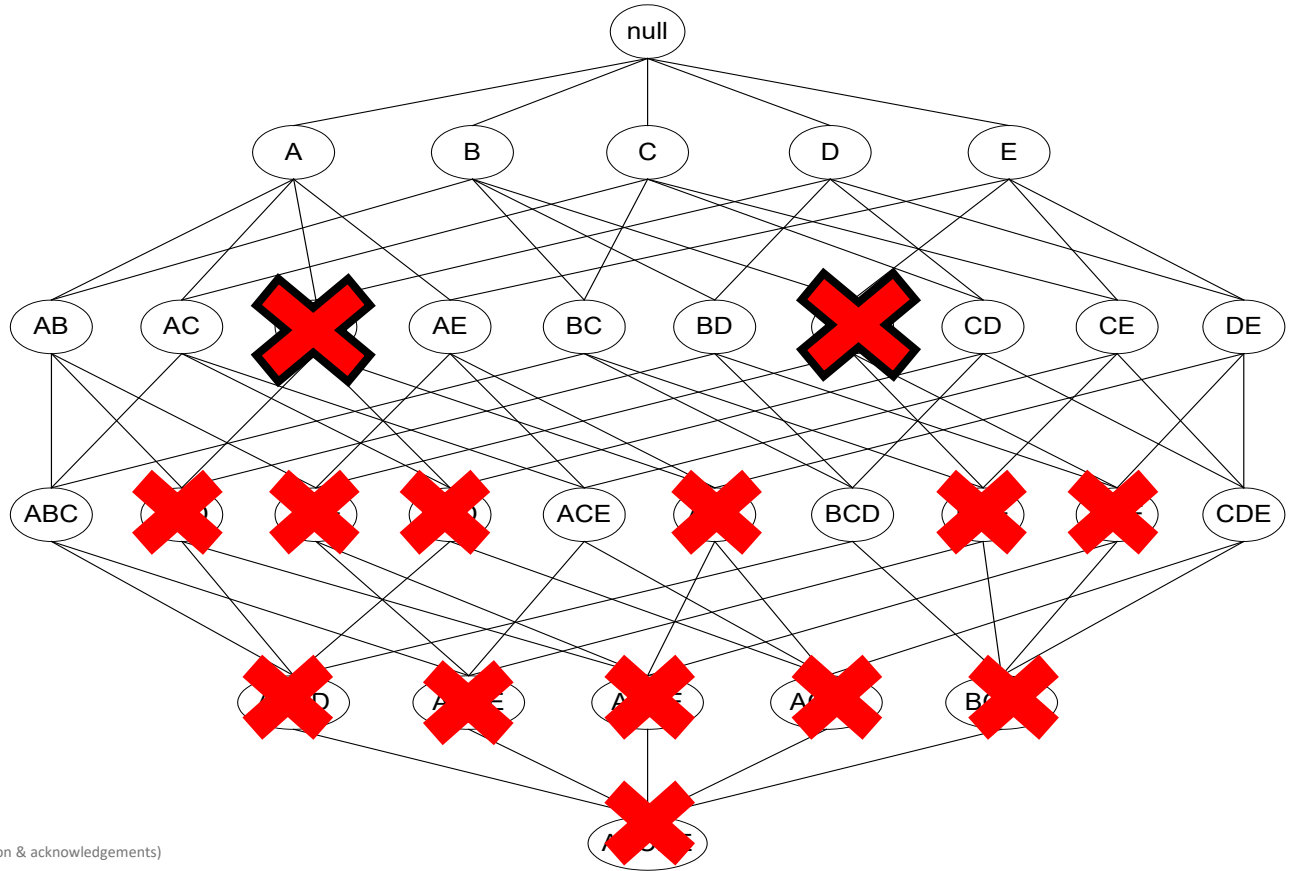
Chair of Process and Data Science

Consider a data set with 5 items



Chair of Process and Data Science

Consider a data set with 5 items

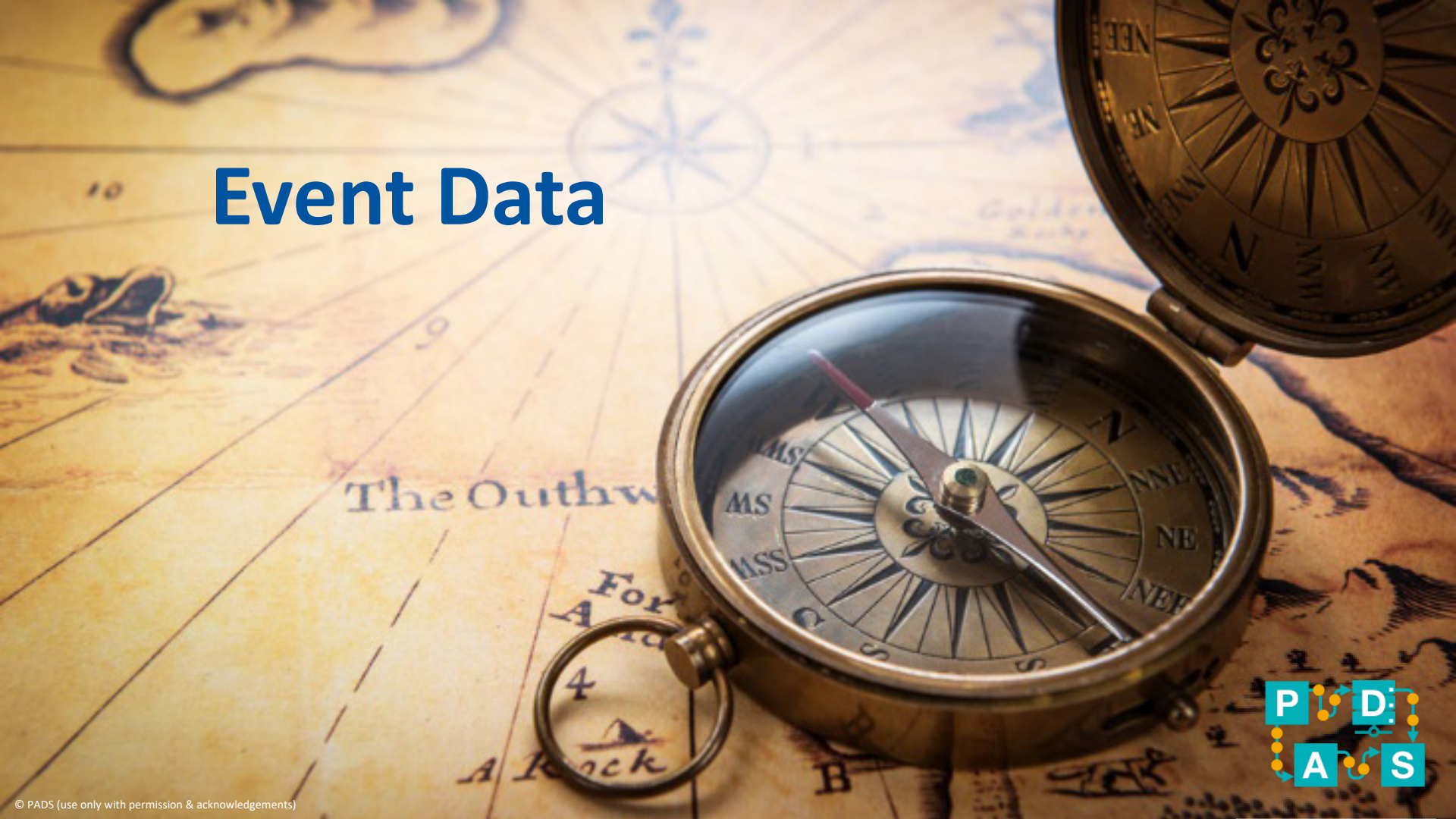


Chair of Process and Data Science

Apriori principle can be used when searching for other “patterns”

- Frequent sequences
- Frequent subgraphs
- Frequent partial orders
- Etc.

Event Data



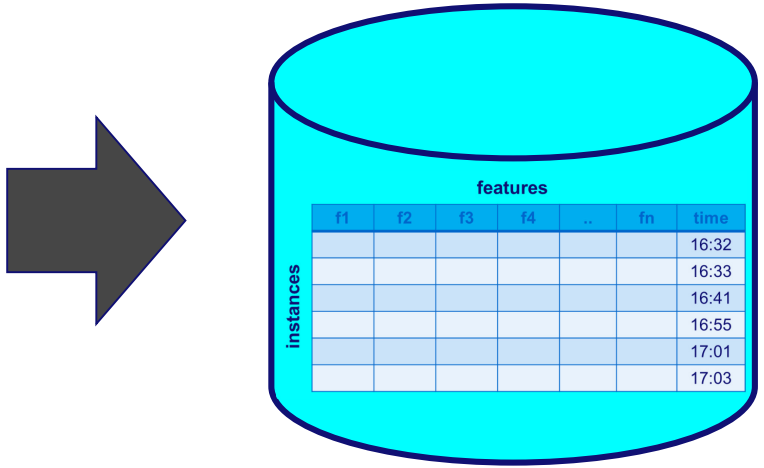
Event data: Discrete timestamped events

features

instances

f1	f2	f3	f4	..	fn	time
						16:32
						16:33
						16:41
						16:55
						17:01
						17:03

Event data are everywhere



Chair of Process and Data Science

Example 1

every row is an event (here: an exam attempt)

student name	course name	exam date	mark
Peter Jones	Business Information systems	16-1-2014	8
Sandy Scott	Business Information systems	16-1-2014	5
Bridget White	Business Information systems	16-1-2014	9
John Anderson	Business Information systems	16-1-2014	8
Sandy Scott	BPM Systems	17-1-2014	7
Bridget White	BPM Systems	17-1-2014	8
Sandy Scott	Process Mining	20-1-2014	5
Bridget White	Process Mining	20-1-2014	9
John Anderson	Process Mining	20-1-2014	8
…	…	…	…

case id | **activity name** | **timestamp** | **other data**

Example 2

order number	activity	timestamp	user	product	quantity
9901	register order	22-1-2014@09.15	Sara Jones	iPhone5S	1
9902	register order	22-1-2014@09.18	Sara Jones	iPhone5S	2
9903	register order	22-1-2014@09.27	Sara Jones	iPhone4S	1
9901	check stock	22-1-2014@09.49	Pete Scott	iPhone5S	1
9901	ship order	22-1-2014@10.11	Sue Fox	iPhone5S	1
9903	check stock	22-1-2014@10.34	Pete Scott	iPhone4S	1
9901	handle payment	22-1-2014@10.41	Carol Hope	iPhone5S	1
9902	check stock	22-1-2014@10.57	Pete Scott	iPhone5S	2
9902	cancel order	22-1-2014@11.08	Carol Hope	iPhone5S	2
…	…	…	…	…	…

case id **activity name** **timestamp** **resource** **other data**

Example 3

patient	activity	timestamp	doctor	age	cost
5781	make X-ray	23-1-2014@10.30	Dr. Jones	45	70.00
5541	blood test	23-1-2014@10.18	Dr. Scott	61	40.00
5833	blood test	23-1-2014@10.27	Dr. Scott	24	40.00
5781	blood test	23-1-2014@10.49	Dr. Scott	45	40.00
5781	CT scan	23-1-2014@11.10	Dr. Fox	45	1200.00
5833	surgery	23-1-2014@12.34	Dr. Scott	24	2300.00
5781	handle payment	23-1-2014@12.41	Carol Hope	45	0.00
5541	radiation therapy	23-1-2014@13.57	Dr. Jones	61	140.00
5541	radiation therapy	23-1-2014@13.08	Dr. Jones	61	140.00
…	…	…	…	…	…

case id | **activity name** | **timestamp** | **resource** | **other data**

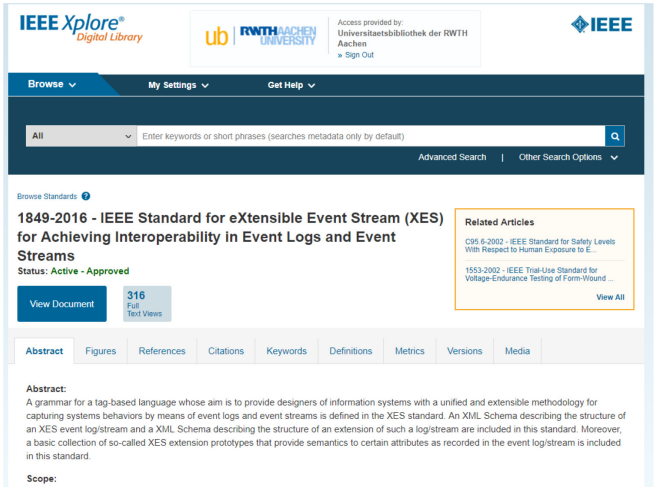
Mandatory attributes for process mining

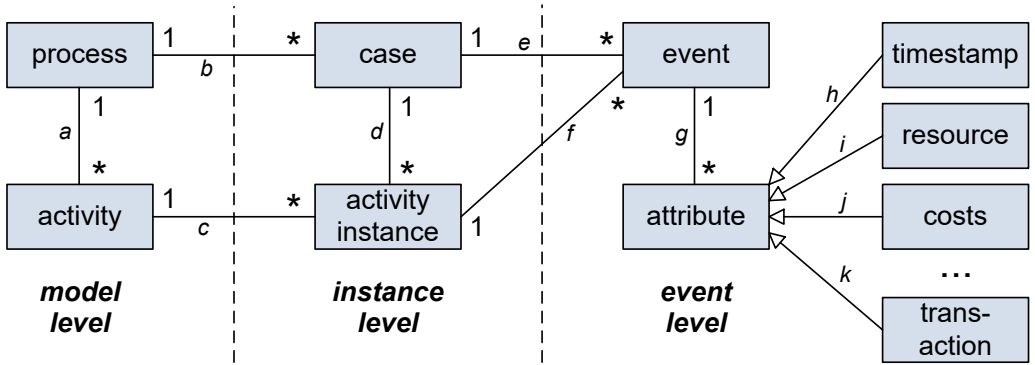
- **Case id**: the “thing” used to group events (customer, order, student, session, device, suitcase, etc.).
- **Activity name**: description of the event, typically an activity, but it could also be an itemset.
- **Timestamp**: the time at which the event takes place

There may be various attributes with a well understood meaning: resource, lifecycle (start, complete, etc.), costs, role, etc. However, events are atomic.

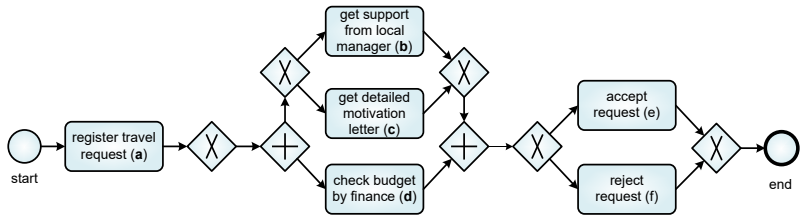
Process mining starts from event data

XES
Extensible Event Stream



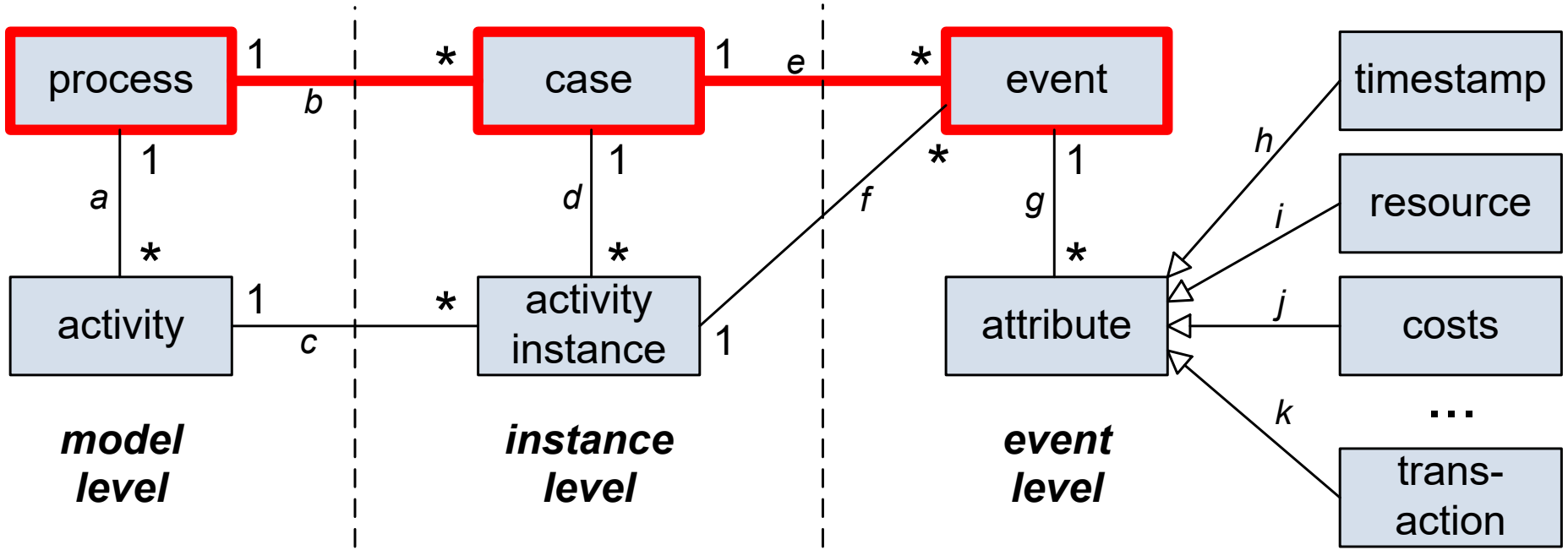


and learns end-to-end process models



Chair of Process and Data Science

Process - case - event



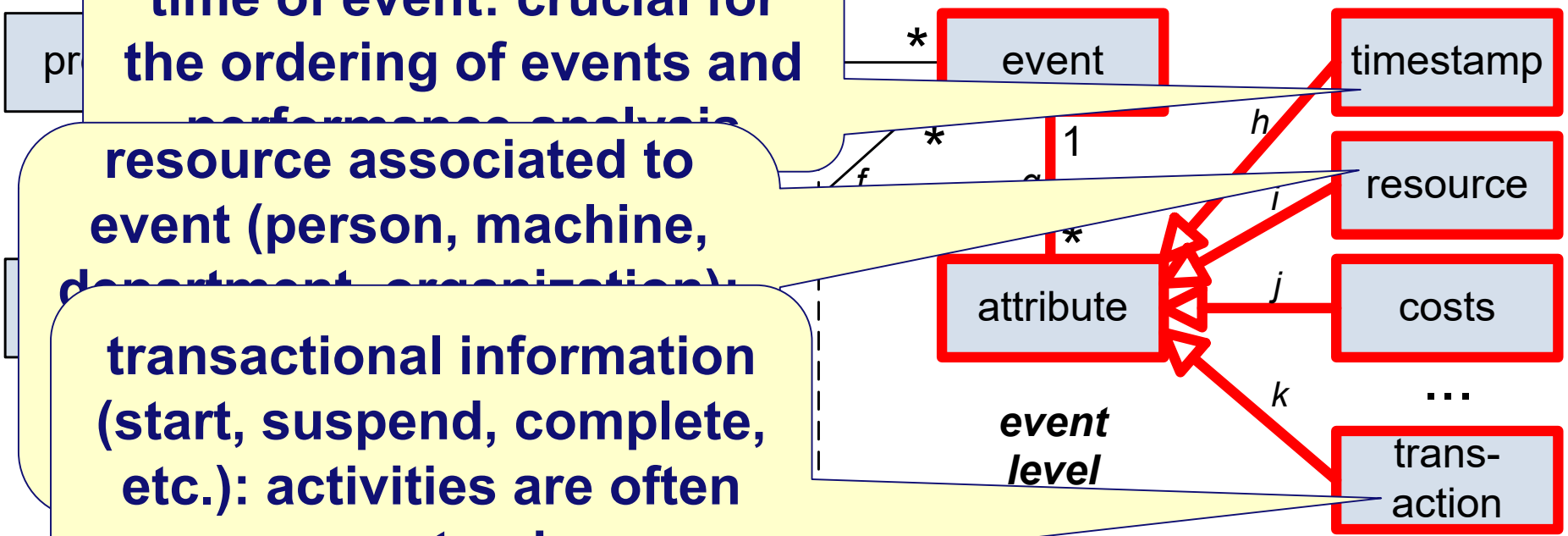
Chair of Process and Data Science

Events have attributes

time of event: crucial for the ordering of events and performance analysis

resource associated to event (person, machine, department, organization)

transactional information (start, suspend, complete, etc.): activities are often non-atomic



Chair of Process and Data Science

Typical abstraction used by process mining algorithms: A multiset of activity sequences

order number	activity	timestamp	user	product	quantity
9901	register order	22-1-2014@09.15	Sara Jones	iPhone5S	1
9902	register order	22-1-2014@09.18	Sara Jones	iPhone5S	2
9903	register order	22-1-2014@09.27	Sara Jones	iPhone4S	1
9901	check stock	22-1-2014@09.49	Pete Scott	iPhone5S	1
9901	ship order	22-1-2014@10.11	Sue Fox	iPhone5S	1
9903	check stock	22-1-2014@10.34	Pete Scott	iPhone4S	1
9901	handle payment	22-1-2014@10.41	Carol Hope	iPhone5S	1
9902	check stock	22-1-2014@10.57	Pete Scott	iPhone5S	2
9902	cancel order	22-1-2014@11.08	Carol Hope	iPhone5S	2
…	…	…	…	…	…

one activity sequence per order

9901 9902

$[\langle \text{register order, check stock, ship order, handle payment} \rangle, \langle \text{register}$

9904

same sequence may appear many times

Typical abstraction used by process mining algorithms: A multiset of activity sequences

patient	activity	timestamp	doctor	age	cost
5781	make X-ray	23-1-2014@10.30	Dr. Jones	45	70.00
5541	blood test	23-1-2014@10.18	Dr. Scott	61	40.00
5833	blood test	23-1-2014@10.27	Dr. Scott	24	40.00
5781	blood test	23-1-2014@10.49	Dr. Scott	45	40.00
5781	CT scan	23-1-2014@11.10	Dr. Fox	45	1200.00
5833	surgery	23-1-2014@12.34	Dr. Scott	24	2300.00
5781	handle payment	23-1-2014@12.41	Carol Hope	45	0.00
5541	radiation therapy	23-1-2014@13.57	Dr. Jones	61	140.00
5541	radiation therapy	23-1-2014@13.08	Dr. Jones	61	140.00
…	…	…	…	…	…

one activity sequence per patient

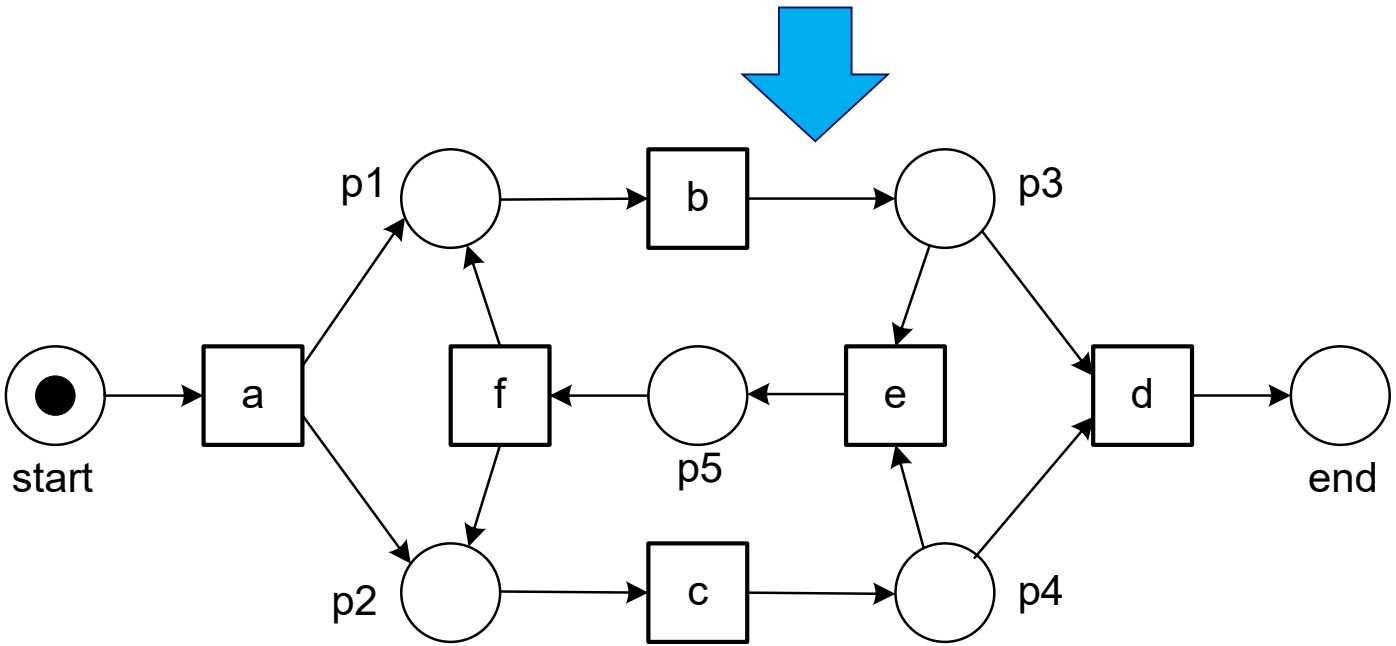
5781 5541

$[\langle$make X–ray, blood test, handle payment$\rangle, \langle$blood test, radiation

same sequence may appear many times

Process discovery (presented in detail in later lectures)

$$L_2 = [\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^4, \langle a,b,c,e,f,b,c,d\rangle^2, \langle a,b,c,e,f,c,b,d\rangle,$$
$$\langle a,c,b,e,f,b,c,d\rangle^2, \langle a,c,b,e,f,b,c,e,f,c,b,d\rangle]$$



Chair of Process and Data Science

Sequential pattern mining

- **Uses a specific type of event data as input.**
- **For example,**
  - Informal: [ab(cd)e, ab(cd)e, a(cd)e, a(bc)(cde)f].
  - Formal:

$$[\langle\{a\},\{b\},\{c,d\},\{e\}\rangle,\langle\{a\},\{b\},\{c,d\},\{e\}\rangle,$$
$$\langle\{a\},\{c,d\},\{e\}\rangle,\langle\{a\},\{b,c\},\{c,d,e\},\{f\}\rangle]$$

- **Formally,** $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$.

Mining sequential patterns

Two beautiful/influential papers

(especially considering that they are almost 25 years old)

Fast Algorithms for Mining Association Rules

Rakesh Agrawal Ramakrishnan Srikant*

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

We consider the problem of discovering association rules between items in a large database of sales transactions. We present two new algorithms for solving this problem that are fundamentally different from the known algorithms. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems. We also show how the best features of the two proposed algorithms can be combined into a hybrid algorithm, called AprioriHybrid. Scale-up experiments show that AprioriHybrid scales linearly with the number of transactions. AprioriHybrid also has excellent scale-up properties with respect to the transaction size and the number of items in the database.

1 Introduction

Progress in bar-code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred to as the *basket* data. A record in such data typically consists of the transaction date and the items bought in the transaction. Successful organizations view such databases as important pieces of the marketing infrastructure. They are interested in instituting information-driven marketing processes, managed by database technology, that enable marketers to develop and implement customized marketing programs and strategies [6].

The problem of mining association rules over basket data was introduced in [4]. An example of such a rule might be that 98% of customers that purchase tires and auto accessories also get automotive services done. Finding all such rules is valuable for cross-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. The databases involved in these applications are very large. It is imperative, therefore, to have fast algorithms for this task.

The following is a formal statement of the problem [4]: Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Associated with each transaction is a unique identifier, called its *TID*. We say that a transaction T *contains* X, a set of some items in $\mathcal{I}$, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if c% of transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ contain $X \cup Y$. Our rules are somewhat more general than in [4] in that we allow a consequent to have more than one item.

Given a set of transactions $\mathcal{D}$, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively. Our discussion is neutral with respect to the representation of $\mathcal{D}$. For example, $\mathcal{D}$ could be a data file, a relational table, or the result of a relational expression.

An algorithm for finding all association rules, henceforth referred to as the *AIS* algorithm, was presented in [4]. Another algorithm for this task, called the *SETM* algorithm, has been proposed in [13]. In this paper, we present two new algorithms, *Apriori* and *AprioriTid*, that differ fundamentally from these algorithms. We present experimental results showing

*Visiting from the Department of Computer Science, University of Wisconsin, Madison.

Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994

Rakesh Agrawal, Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499

1 year, both using the same Apriori principle

Mining Sequential Patterns

Rakesh Agrawal Ramakrishnan Srikant*

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

We are given a large database of customer transactions, where each transaction consists of customer-id, transaction time, and the items bought in the transaction. We introduce the problem of mining sequential patterns over such databases. We present three algorithms to solve this problem, and empirically evaluate their performance using synthetic data. Two of the proposed algorithms, AprioriSome and AprioriAll, have comparable performance, albeit AprioriSome performs a little better when the minimum number of customers that must support a sequential pattern is low. Scale-up experiments show that both AprioriSome and AprioriAll scale linearly with the number of customer transactions. They also have excellent scale-up properties with respect to the number of transactions per customer and the number of items in a transaction.

1 Introduction

Database mining is motivated by the decision support problem faced by most large retail organizations. Progress in bar-code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred to as the *basket* data. A record in such data typically consists of the transaction date and the items bought in the transaction. Very often, data records also contain customer-id, particularly when the purchase has been made using a credit card or a frequent-buyer card. Catalog companies also collect such data using the orders they receive.

We introduce the problem of mining *sequential* patterns over this data. An example of such a pattern is that customers typically rent "Star Wars", then "Empire Strikes Back", and then "Return of the Jedi". Note that these rentals need not be consecutive. Customers who rent some other videos in between also support this sequential pattern. Elements of a sequential pattern need not be simple items. "Fitted Sheet and flat sheet and pillow cases", followed by "comforter", followed by "drapes and ruffles" is an example of a sequential pattern in which the elements are sets of items.

Problem Statement We are given a database $\mathcal{D}$ of customer transactions. Each transaction consists of the following fields: customer-id, transaction-time, and the items purchased in the transaction. No customer has more than one transaction with the same transaction-time. We do not consider quantities of items bought in a transaction: each item is a binary variable representing whether an item was bought or not.

An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets. Without loss of generality, we assume that the set of items is mapped to a set of contiguous integers. We denote an itemset i by $(i_1 i_2 \ldots i_m)$, where i_j is an item. We denote a sequence s by $\langle s_1 s_2 \ldots s_n \rangle$, where s_j is an itemset.

A sequence $\langle a_1 a_2 \ldots a_n \rangle$ *is contained in* another sequence $\langle b_1 b_2 \ldots b_m \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, ..., $a_n \subseteq b_{i_n}$. For example, the sequence $\langle (3)\ (4\ 5)\ (8) \rangle$ is contained in $\langle (7)\ (3\ 8)\ (9)\ (4\ 5\ 6)\ (8) \rangle$, since $(3) \subseteq (3\ 8)$, $(4\ 5) \subseteq (4\ 5\ 6)$ and $(8) \subseteq (8)$. However, the sequence $\langle (3)\ (5) \rangle$ is not contained in $\langle (3\ 5) \rangle$ (and vice versa). The former represents items 3 and 5 being bought one after the other, while the latter represents items 3 and 5 being bought together. In a set of sequences, a sequence s is *maximal* if s is not contained in any other sequence.

All the transactions of a customer can together be viewed as a sequence, where each transaction corresponds to a set of items, and the list of

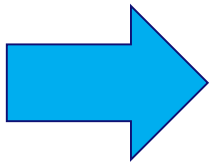
*Also Department of Computer Science, University of Wisconsin, Madison.

Rakesh Agrawal, Ramakrishnan Srikant: Mining Sequential Patterns. ICDE 1995: 3-14

Chair of Process and Data Science

Input: Multiset of sequences of itemsets

Customer Id	TransactionTime	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) ⟩

Multiset of sequences of itemsets

Taken from Rakesh Agrawal, Ramakrishnan Srikant: Mining Sequential Patterns. ICDE 1995: 3-14

Sequential Patterns with support > 25%
⟨ (30) (90) ⟩
⟨ (30) (40 70) ⟩

Input: Multiset of sequences of itemsets

- $\mathcal{I}$ is the **set of all items**. An **itemset** is a nonempty set of items, e.g., $i = \{i_1, i_2, \ldots, i_m\} \subseteq \mathcal{I}$ with $m \geq 1$.
- A **sequence** is a nonempty sequence of itemsets, e.g., $s = \langle s_1, s_2, \ldots, s_n \rangle \in \left(\mathcal{P}(\mathcal{I})\right)^*$ with $n \geq 1$.
- A **dataset** D is a multiset of sequences.
- Technically, $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$.

($\mathfrak{B}$ is the multiset and $\mathcal{P}$ is the powerset operator)

Example input: $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$

$[\langle\{a\},\{a,b\},\{b,c\},\{c\}\rangle, \langle\{a\},\{a\},\{a,b\},\{b,c\},\{c\}\rangle,$
$\langle\{a\},\{a\},\{d\},\{a\}\rangle,\langle\{a,b\},\{a,b\},\{a,b\},\{a,b\}\rangle,$
$\langle\{a,b\},\{b,c\},\{c,d,e\}\rangle,\langle\{a\},\{a,b\},\{b,c\},\{c\}\rangle]$

Informal short-hand notation:

[a(ab)(bc)c,aa(ab)(bc)c,aaaa,(ab)(ab)(ab)(ab),
(ab)(bc)(cde),a(ab)(bc)c]

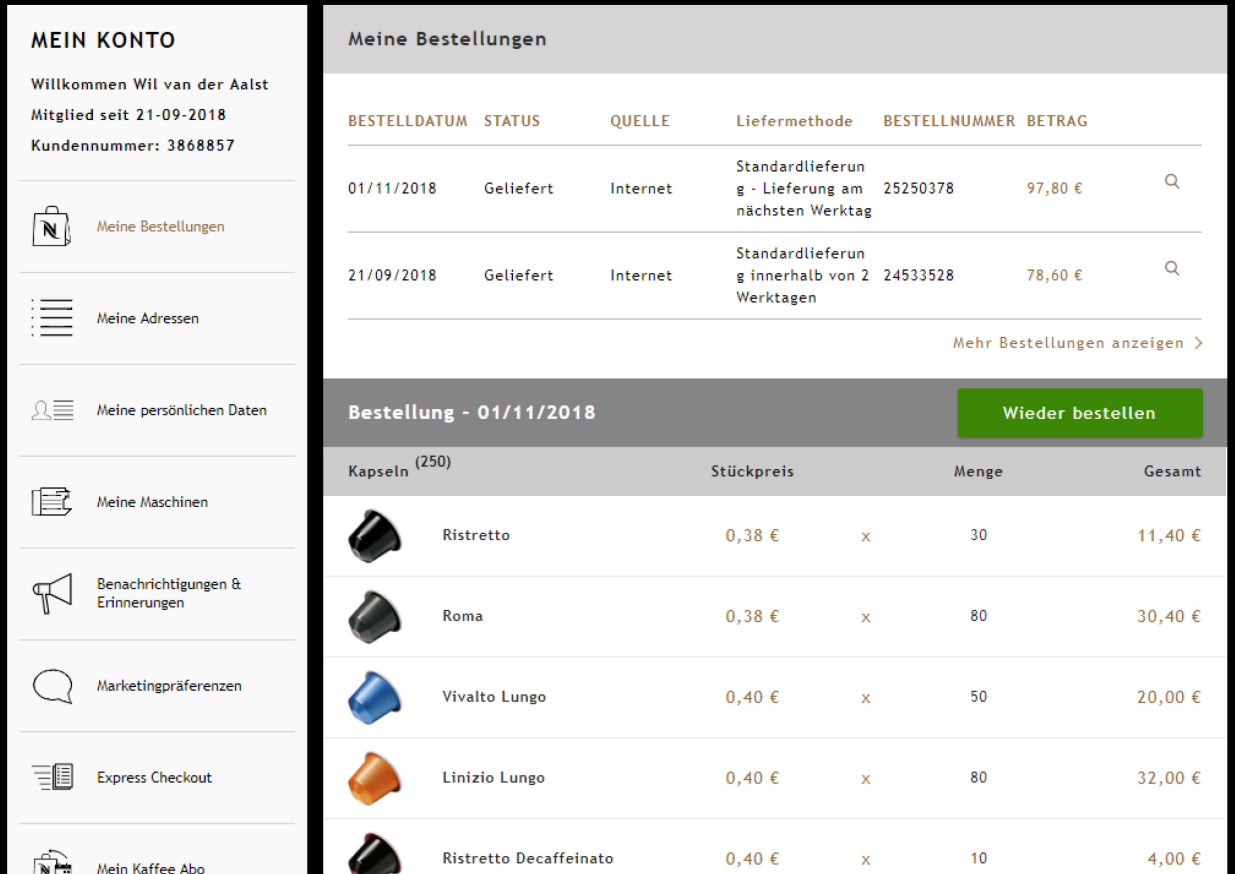
Einkaufswagen

	Preis	Menge
 **Philips Hue White E27 LED Lampe Erweiterung, dimmbar, warmweißes Licht, steuerbar via App, kompatibel mit Amazon Alexa (Echo, Echo Dot)** Auf Lager. ✓prime Dies ist ein Geschenk Erfahren Sie mehr Löschen | Auf die Merkliste	EUR 16,99	1
 **Philips Hue Bridge, zentrales, intelligentes Steuerelement des Hue Systems** Auf Lager. ✓prime Dies ist ein Geschenk Erfahren Sie mehr Löschen | Auf die Merkliste	EUR 46,95	1
 **Philips Hue White E27 LED Lampe Doppelpack, dimmbar, warmweißes Licht, steuerbar via App, kompatibel mit Amazon Alexa (Echo, Echo Dot)** Auf Lager. ✓prime Dies ist ein Geschenk Erfahren Sie mehr Löschen | Auf die Merkliste	EUR 27,99	1
 **Philips Hue White und Color Ambiance E27 LED Lampe Starter Set, drei Lampen** Auf Lager. ✓prime Dies ist ein Geschenk Erfahren Sie mehr Löschen | Auf die Merkliste	EUR 134,90	1

$$D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$$

Summe (4 Artikel): EUR 226,83

$$D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$$



$$D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$$

Goal: Find frequent sequential patterns

- Given a dataset $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$, i.e., a multiset of sequences of itemsets, find all frequent sequential patterns.
- Sequential patterns are of the same form $p \in \left(\mathcal{P}(\mathcal{I})\right)^*$, i.e., sequences of itemsets.
- The support of a sequential pattern p is the fraction (or absolute number) of sequences in D that supports pattern p (i.e., is contained).

Containment

- Let $a = \langle a_1, a_2, \dots, a_n \rangle \in \left(\mathcal{P}(\mathcal{I})\right)^*$ and $b = \langle b_1, b_2, \dots, b_m \rangle \in \left(\mathcal{P}(\mathcal{I})\right)^*$ be two itemset sequences.
- a is contained in b if there exist integers $i_1 < i_2 < \dots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$.
- $a \sqsubseteq b$ if a is contained in b.
- If $a \sqsubseteq b$, then a is a subsequence of b and b is a supersequence of a.

Examples

- $\langle \{a\}, \{a, b\}, \{b, c\}, \{c\} \rangle \sqsubseteq \langle \{a\}, \{a\}, \{a, b, c\}, \{b, c\}, \{b, c\}, \{a, c\} \rangle$
- $\langle \{a\}, \{a, b\}, \{b, c\}, \{c\} \rangle \not\sqsubseteq \langle \{a\}, \{a, b, c\}, \{b, d\}, \{b, e\}, \{a, c\} \rangle$

Informal notation:

- **a(ab)(bc)c $\sqsubseteq$ aa(abc)(bc)(bc)(ac)**
- **a(ab)(bc)c $\not\sqsubseteq$ a(abc)(bd)(be)(ac)**

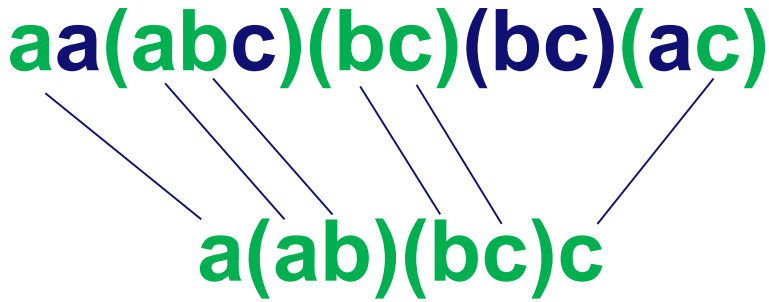
$\langle a_1, a_2, \ldots, a_n \rangle \sqsubseteq \langle b_1, b_2, \ldots, b_m \rangle$ iff there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Examples

- $\langle \{a\}, \{a, b\}, \{b, c\}, \{c\} \rangle \sqsubseteq \langle \{a\}, \{a\}, \{a, b, c\}, \{b, c\}, \{b, c\}, \{a, c\} \rangle$
- $\langle \{a\}, \{a, b\}, \{b, c\}, \{c\} \rangle \not\sqsubseteq \langle \{a\}, \{a, b, c\}, \{b, d\}, \{b, e\}, \{a, c\} \rangle$

Informal notation:

- a(ab)(bc)c ⊑ aa(abc)(bc)(bc)(ac)
- a(ab)(bc)c ⋢ a(abc)(bd)(be)(ac)



multiple mappings possible

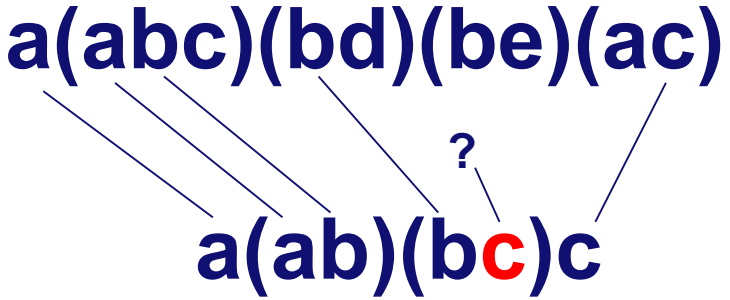
$\langle a_1, a_2, \ldots, a_n \rangle \sqsubseteq \langle b_1, b_2, \ldots, b_m \rangle$ iff there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Examples

- $\langle\{a\},\{a,b\},\{b,c\},\{c\}\rangle \sqsubseteq \langle\{a\},\{a\},\{a,b,c\},\{b,c\},\{b,c\},\{a,c\}\rangle$
- $\langle\{a\},\{a,b\},\{b,c\},\{c\}\rangle \not\sqsubseteq \langle\{a\},\{a,b,c\},\{b,d\},\{b,e\},\{a,c\}\rangle$

Informal notation:

- **a(ab)(bc)c ⊑ aa(abc)(bc)(bc)(ac)**
- **a(ab)(bc)c ⋢ a(abc)(bd)(be)(ac)**



$\langle a_1, a_2, \ldots, a_n\rangle \sqsubseteq \langle b_1, b_2, \ldots, b_m\rangle$ iff there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Questions

- (ab)(bc) $\sqsubseteq$ (bc)(ab) ?
- ab $\sqsubseteq$ a(ac)(bc)c ?
- aa(ab)(bc) $\sqsubseteq$ (ab)(ace)(bce)(ab) ?
- (abc)ef $\sqsubseteq$ (ab)(bc)(ef)f ?
- (abc)ef $\sqsubseteq$ (ab)(bc)(abcd)(ef)f ?

$\langle a_1, a_2, \dots, a_n \rangle \sqsubseteq \langle b_1, b_2, \dots, b_m \rangle$ iff there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Answers

- (ab)(bc) $\not\sqsubseteq$ (bc)(ab) (incompatible order)
- ab $\sqsubseteq$ a(ac)(bc)c
- aa(ab)(bc) $\not\sqsubseteq$ (ab)(ace)(bce)(ab)
  ((ab) cannot be mapped without also handling aa)
- (abc)ef $\not\sqsubseteq$ (ab)(bc)(ef)f (no match for (abc))
- (abc)ef $\sqsubseteq$ (ab)(bc)(abcd)(ef)f

$\langle a_1, a_2, \ldots, a_n \rangle \sqsubseteq \langle b_1, b_2, \ldots, b_m \rangle$ iff there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Support (relative)

- The support of a sequence pattern $p \in \left(\mathcal{P}(\mathcal{I})\right)^*$ given a dataset $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$ is the fraction of sequences in D that are supersequences of p.
- $support(p) = |[s \in D | p \sqsubseteq s]|/|D|$ (relative)
- Minimum support threshold min_sup: lower bound for $support(p)$.

Support (absolute)

- $support\_count(p) = |[s \in D | p \sqsubseteq s]|$ (absolute, also called frequency or count)
- Minimum support count threshold: lower bound for $support\_count(p)$.

Questions

- D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
- What is the $support\_count(p)$ for
  - p=a
  - p=ab
  - p=(ab)
  - p=(ab)c
  - p=(ab)(bd)
  - p=ab(cd)

Questions

- D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
- What is the $support\_count(p)$ for
  - p=a : 4 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
  - p=ab : 2 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
  - p=(ab) : 3 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
  - p=(ab)c : 2 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
  - p=(ab)(bd) : 0 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]
  - p=ab(cd) : 1 - D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)]

AprioriAll Algorithm

Brute force approach

- Let k be the length of the longest sequence in $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$ and q the largest itemset.
- Generate all sequence patterns $p \in \left(\mathcal{P}(\mathcal{I})\right)^*$ having a length $\leq k$ and itemsets of size $\leq q$. This number is finite.
- Compute the support of each candidate pattern.
- Return all that have a support higher than min_sup.
- Obviously this is very expensive.

A smarter approach based on Apriori

- First described in Rakesh Agrawal, Ramakrishnan Srikant: Mining Sequential Patterns. ICDE 1995: 3-14.
- Just like for frequent itemsets: Avoid testing hopeless candidates.
- If $p_1 \sqsubseteq p_2$ (p_1 is a subsequence of p_2), then p_2 cannot be frequent if p_1 is not frequent.
  - $support(p_1) \geq support(p_2)$ if $p_1 \sqsubseteq p_2$
  - if $p_1 \sqsubseteq p_2$ and $support(p_1) < min\_sup$, then $support(p_2) < min\_sup$

Step 1: Determine all litemsets L

- $L = \{i \subseteq \mathcal{I} | support(\langle i \rangle) \geq min\_sup\}$ are all itemsets that appear in a sufficient number of sequences.
- These itemsets are called litemsets.
- For example, for D=[abcd,(abcd),(ab)(cd),(ab)(bc)(cd)] and $min\_sup = 0.7$ the following itemsets are frequent: a (support = 4/4), b (support = 4/4), c (support = 4/4), d (support = 4/4), (ab) (support = 3/4), (cd) (support = 3/4).
- To determine all litemsets, we can use a variant of the original Apriori algorithm (the only difference is that support is counted per sequence and not per itemset).

Step 2: Preprocess dataset

- We only need to consider litemsets (i.e., L): There cannot be any patterns that involve other itemsets.
- Frequent sequence patterns must be of the form L^*!
- The set $L_1 = \{\langle i \rangle | i \in L\}$ is the set of all frequent sequence patterns of length 1.
- $L_k \subseteq L^*$ is the set of all frequent sequence patterns of length k (to be computed).

Transformation (only for optimization, using formal notation)

- Transform $D \in \mathfrak{B}\left(\left(\mathcal{P}(\mathcal{I})\right)^*\right)$ into $D_T \in \mathfrak{B}\left(\left(\mathcal{P}(L)\right)^*\right)$.
- Each sequence is now described by a sequence of sets of litemsets.
- Let $L = \{\{a\}, \{b\}, \{c\}, \{a, b\}\}$ and $D = [\langle\{a, c\}, \{a, b, c\}\rangle, \langle\{c\}, \{a, c\}\rangle, \dots]$.
- $D_T = [\langle\{\{a\}\{c\}\}, \{\{a\}, \{b\}, \{c\}, \{a, b\}\}\rangle, \langle\{\{c\}\}, \{\{a\}, \{c\}\}\rangle, \dots]$
- The new representation makes it very easy to test whether a sequence pattern is supported by a sequence in the data set.

Step 3: Generate set of candidate sequences

- Assume we have L_{k-1}, the set of all frequent sequence patterns of length $k-1$. Recall that $L_1 = \{\langle i \rangle | i \in L\}$.
- C_k is the set of all candidate sequences obtained by taking two sequences from L_{k-1} where the first $k-1$ are the same.

Step 4: Prune the set of candidate sequences

- For all $\mathbf{c} \in C_k$.
  - Consider all subsequences of $\mathbf{c}$ of length $k-1$.
  - If one of these subsequences is not in L_{k-1}, then remove $\mathbf{c}$ from C_k.

Step 5: Test all candidate sequences

- For each transformed sequence $s \in D_T$: Increment the count of $c \in C_k$ if c is contained in s.
- Remove all candidates $c \in C_k$ that do no meet the threshold and the result is L_k.
- $L_k = \{c \in C_k | support(c) \geq min\_sup\}$.
- Increment k and goto Step 3 until $L_k = \emptyset$.
- $\bigcup_k L_k$ is the set of all frequent sequence patterns

Step 6 (optional): Remove non-maximal patterns

- An sequence s is a maximal sequence in data set D if s is frequent, and there is no other supersequence s' that is also frequent ($s \sqsubset s'$).
- It is possible to keep only the maximal sequences. However, support information for the subsequences will be lost (these may be more frequent).

Other sequential pattern mining approaches

From Carl Mooney, John Roddick, Sequential pattern mining: approaches and algorithms, ACM Comput. Surv.,45(2):1-39, 2013.

Table VII. A summary of Apriori-based algorithms.

Algorithm Name	Author	Year	Notes
Candidate Generation: Horizontal Database Format			
Apriori (All, Some, Dynamic Some)	[Agrawal and Srikant]	1995	
Generalised **S**equential **P**atterns (GSP)	[Srikant and Agrawal]	1996	Max/Min Gap, Window, Taxonomies
PSP	[Masseglia et al.]	1998	Retrieval optimisations
Sequential **P**attern m**I**ning with **R**egular expres**I**on cons**T**raints (SPIRIT)	[Garofalakis et al.]	1999	Regular Expressions
Maximal **F**requent **S**equences (MFS)	[Zhang et al.]	2001	Based on GSP, uses Sampling
Regular **E**xpression-**H**ighly **A**daptive **C**onstrained **L**ocal **E**xtractor (RE-Hackle)	[Albert-Lorincz and Boulicaut]	2003	Regular Expressions, similar to SPIRIT
Maximal **S**equential **P**atterns using **S**ampling (MSPS)	[Luo and Chung]	2004	Sampling
Candidate Generation: Vertical Database Format			
Sequential **PA**ttern **D**iscovery using **E**quivalence classes (SPADE)	[Zaki]	2001	Equivalence Classes
Sequential **PA**ttern **M**ining (SPAM)	[Ayres et al.]	2002	Bitmap representation
LAst **P**osition **IN**duction (LAPIN)	[Yang and Kitsuregawa]	2004	Uses last position
Cache-based **C**onstrained **S**equence **M**iner (CCSM)	[Orlando et al.]	2004	k-way intersections, cache
Indexed **B**it **M**ap (IBM)	[Savary and Zeitouni]	2005	Bit Map, Sequence Vector, Index, NB table
LAst **P**osition **IN**duction **S**equential **PA**ttern **M**ining (LAPIN-SPAM)	[Yang and Kitsuregawa]	2005	Uses SPAM, Uses last position

Table IX. A summary of pattern growth algorithms.

Algorithm Name	Author	Year	Notes
Pattern Growth			
FREquE**n**t pattern-projected **S**equential **PA**tter**N** mining (FreeSpan)	[Han et al.]	2000	Projected sequence database
PREFIX-projected **S**equential **PA**tter**N** mining (PrefixSpan)	[Pei et al.]	2001	Projected prefix database
Sequential pattern mining with **L**ength-decreasing su**P**port (SLPMiner)	[Seno and Karypis]	2002	Length-decreasing support

Table XI. A summary of temporal sequence algorithms.

Algorithm Name	Author	Year	Notes
Candidate Generation: Episodes			
WINEPI	[Mannila et al.]	[1995], 1996	State automata, Window
WINEPI, MINEPI	[Mannila et al.]	1997	State automata, Window, Maximal
Pattern Directed Mining	[Guralnik et al.]	1998	Pattern language
Event-Oriented Patterns	[Sun et al.]	2003	Target events
Pattern Growth: Episodes			
PROjected **W**indow **L**ists (PROWL)	[Huang et al.]	2004	Projected window lists

Chair of Process and Data Science

Extensions

Association rules based on frequent sequences

- **Frequent sequence patterns can be split in an “if” and “then” part (similar to “normal” association rules).**
- $\langle \{beer\}, \{red, white\} \rangle \Rightarrow \langle \{beer\}, \{red, white\}, \{wodka\} \rangle$
- $\langle \{beer\}, \{beer\} \rangle \Rightarrow \langle \{beer\}, \{beer\}, \{beer\}, \{beer\}, \{beer\} \rangle$
- **Many variants possible.**

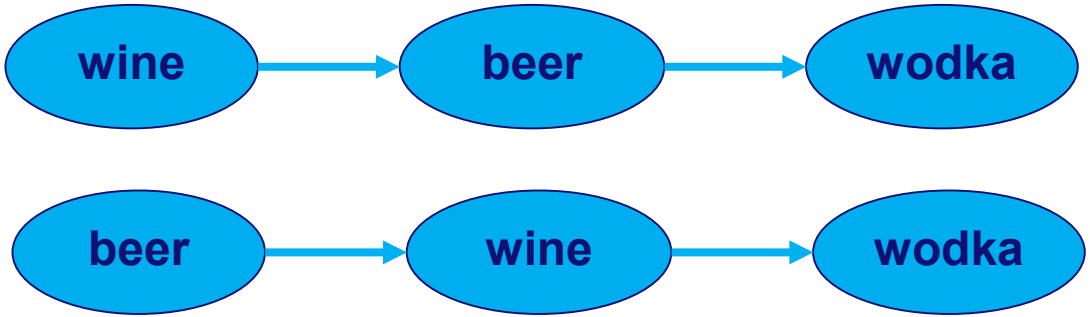
How interesting are frequent sequences?

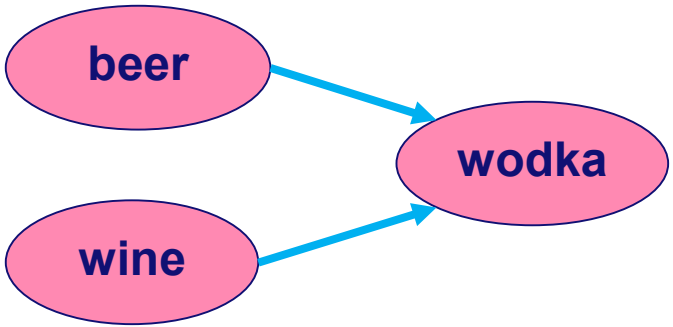
- For any technique that identifies patterns, it is important to filter out the less interesting ones.
- One can look at things like correlations and base frequencies to decide how surprising sequences or sequence-based rules are.
- It is also possible to add further constraints.

Additional constraints

- Item constraints (only consider sequences that include or exclude a set of items).
- Length constraints (only consider patterns of a given size).
- Time constraints (only consider patters that occur in a short timeframe). This includes gap and duration constraints.
- Regular expression constraints (only consider patterns that satisfy a regular expression or temporal constraint).

Episode mining





Rather than looking for sequences we look for embedded partial orders.

Reminder: Data science is like a making cocktail. Mix the proper ingredients in the right way!

Python
Visualization
Decision trees
Regression
Support vector machines
Neural networks
Evaluation
Clustering
Frequent items sets
Association rules
Sequence mining
Process mining
Process discovery
Conformance checking
Text mining
Preprocessing
Visual analytics
Encryption
Anonymization
Big data infra
Distribution

Sequence mining

↑

Clustering

↑

Visualization

↑

Preprocessing

Python
Visualization
Decision trees
Regression
Support vector machines
Neural networks
Evaluation
Clustering
Frequent items sets
Association rules
Sequence mining
Process mining
Process discovery
Conformance checking
Text mining
Preprocessing
Visual analytics
Encryption
Anonymization
Big data infra
Distribution

Visualization

Support vector machines

Clustering

Sequence mining

Preprocessing

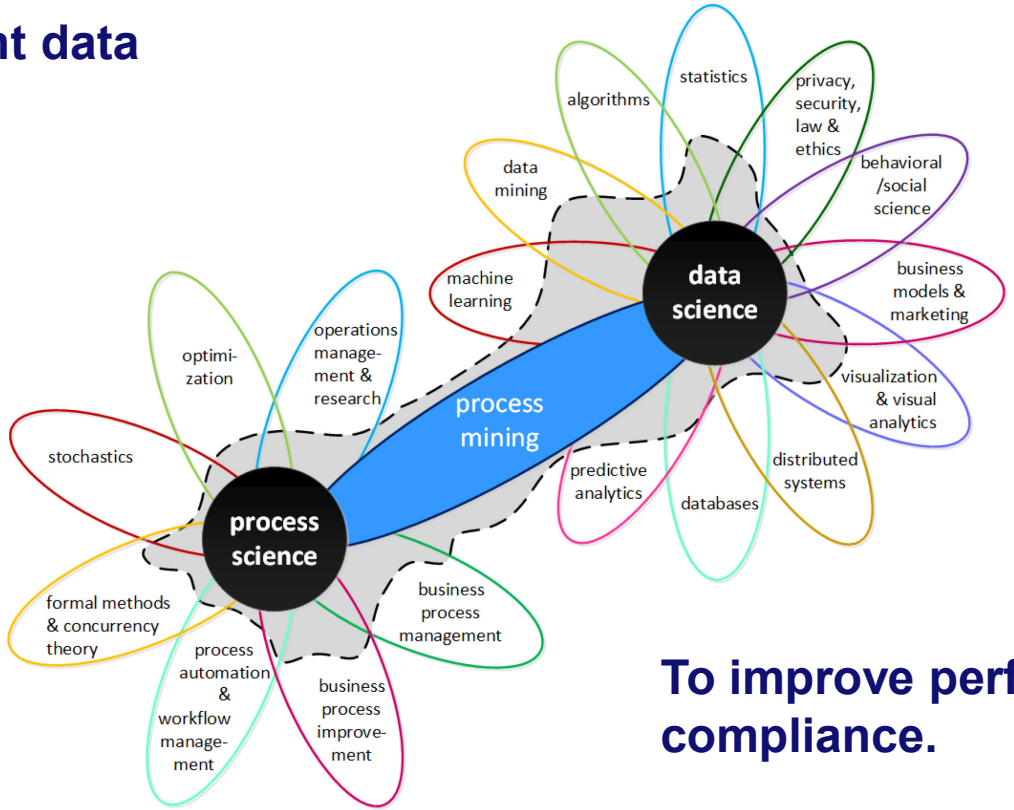
Conclusion

Short summary of lecture

- Event data are everywhere!
- Many questions related to event data!
- Sequence mining looks for local patterns.
- Apriori-style approach can be used (builds on frequent itemsets).

Next process mining

Starting point: Event data



To improve performance and compliance.

Chair of Process and Data Science

Sequence mining versus process mining

Sequence mining	Process mining
Focus on local patterns	Focus on end-to-end processes
Focus on itemsets	Focus on activities
Low-level patterns	Higher-level behaviors (e.g., concurrency and choice).
Specific event data (itemset based)	General event data (case, activity, resource, lifecycle, etc.)
…	…

#	Lecture	date	day
Lecture 1	Introduction	10/10/2018	Wednesday
Lecture 2	Crash Course in Pytho		
Instruction 1	*Python*		
Lecture 3	Basic data visualisatio		
Lecture 4	Decision trees		
Instruction 2	*Decision trees and da*		
Lecture 5	Regression		
Lecture 6	Support vector machi		
Instruction 3	*Regression and suppo*		
Lecture 7	Neural networks (1/2		
Instruction 4	*Neural networks and*		
Lecture 8	Neural networks (2/2		
Lecture 9	Evaluation of supervi		
Instruction 5	*Neural networks and*		
Lecture 10	Clustering		
Lecture 11	Frequent items sets	15/11/2018	Thursday
Lecture 12	Association rules	21/11/2018	Wednesday
Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association ru*	*23/11/2018*	*Friday*
Lecture 14	Process mining (unsupervised)	28/11/2018	Wednesday
Lecture 15	Process mining (supervised)	29/11/2018	Thursday
Instruction 7	*Process mining and sequence mining*	*30/11/2018*	*Friday*
Lecture 16	Text mining (1/2)	05/12/2018	Wednesday
Instruction 8	*Text mining and process mining*	*06/12/2018*	*Thursday !!*
Lecture 17	Text mining (2/2)	12/12/2018	Wednesday
Lecture 18	Data preprocessing, data quality, binning, etc	13/12/2018	Thursday
Lecture 19	Visual analytics & information visualization	19/12/2018	Wednesday
backup		20/12/2018	Thursday
Instruction 9	*Text mining, preprocessing and visualization*	*21/12/2018*	*Friday*
Lecture 20	Responsible data science (1/2)	09/01/2019	Wednesday
Lecture 21	Responsible data science (2/2)	10/01/2019	Thursday
Instruction 10	*Responsible data science*	*11/01/2019*	*Friday*
Lecture 22	Big data (1/2)	16/01/2019	Wednesday
Lecture 23	Big data (2/2)	17/01/2019	Thursday
Instruction 11	*Big data*	*18/01/2019*	*Friday*
Lecture 24	Closing	23/01/2019	Wednesday
backup		24/01/2019	Thursday
Instruction 12	*Example exam questions*	*25/01/2018*	*Friday*
backup		30/01/2019	Wednesday
backup		31/01/2019	Thursday
extra	*Question hour*	*01/02/2019*	*Friday*

Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association ru*	*23/11/2018*	*Friday*
Lecture 14	Process mining (unsupervised)	28/11/2018	Wednesday
Lecture 15	Process mining (supervised)	29/11/2018	Thursday
Instruction 7	*Process mining and sequence mining*	*30/11/2018*	*Friday*
Lecture 16	Text mining (1/2)	05/12/2018	Wednesday
Instruction 8	*Text mining and process mining*	*06/12/2018*	*Thursday !!*