

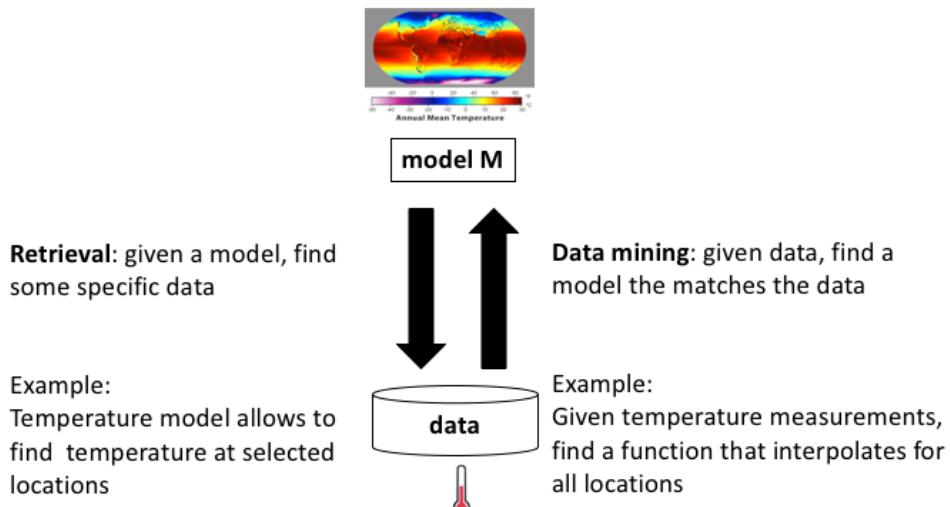
Part 2: Data Mining

Overview

1. Data Mining – Introduction
2. Association Rule Mining
3. Clustering
4. Classification
5. Mining Social Graphs
6. Classification Methodology
7. Document Classification
8. Recommender Systems

1. DATA MINING - INTRODUCTION

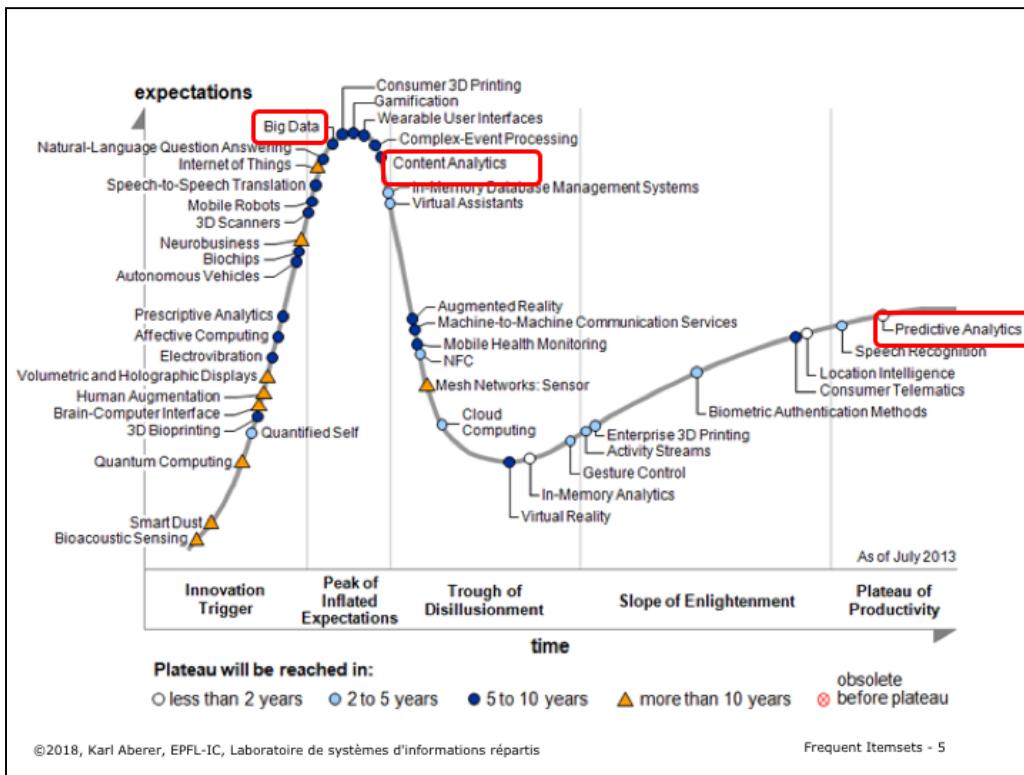
Information Management Tasks



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

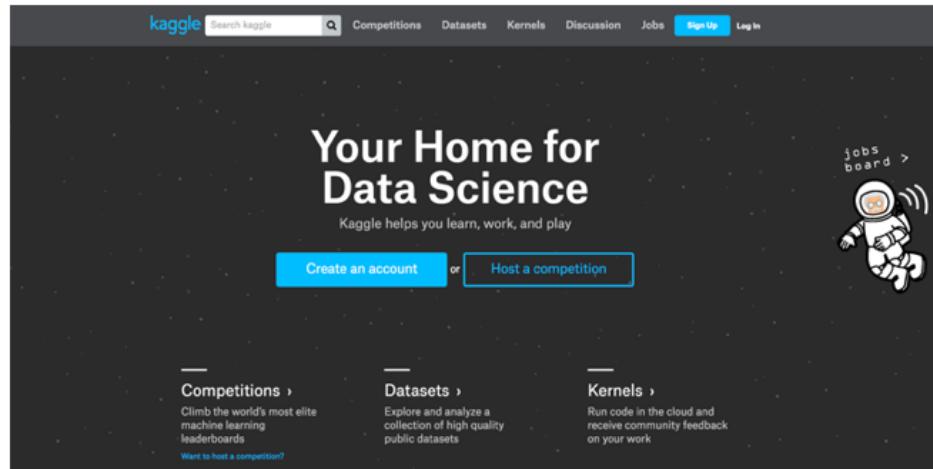
Frequent Itemsets - 4

Since the central role of an information system is to create a model of reality based on data, the key information management tasks are related to the interplay between data and models. We can identify two directions for this interplay: from models to data, and from data to models. From models to data is commonly what we understand by **retrieval**, or information retrieval. Given a model of reality we would like to learn about specific aspects of reality. If we have a model of the temperature distribution in the world, we would like to retrieve the temperature at a specific location or a global average temperature. For Web search we would use a model of how search terms provided by user to a Web search engine relate to documents considered as being relevant by the user, to retrieve the results of a user query. Going from data to models is what we commonly understand as **data mining**. Often we find big data collections for which we do not have a proper or only incomplete interpretation. For example, we might have temperature measurements at given points, but do not understand the correlations among those measurements or the values at locations without measurements. If we have large document collections, we do not understand which are the topics that are covered by those documents. Data mining deals with the problem of providing algorithms that reveal hidden structures in data in order to create new models. Both information retrieval and data mining will be central topics that will be covered in this course.



Data mining (often also called Data Analytics or Predictive Analytics), is one of the most important developments in computer science of the past 25 years. The notion was coined in 1990 by Gregory Piatetsky-Shapiro, who founded at the same time the KDD conference series. Since then it has continuously evolved and wide-spread used in industry and government organizations, as this Gartner hype cycle chart illustrates.

Kaggle



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 6

Data Mining

Data is gathered at an increasing speed and volume
(size doubles each year – Big Data)



4 new petabytes per day



300 hours of video uploaded every minute



1 billion tweets each 48 hours (1st billion took 3 years)



12 EB (1 billion GB) of storage in Utah

The growing use of information systems in many domains produces rapidly growing data collections in business, science and on the Web. Only recently the rate at which digital data is produced started exceeding the rate at which storage space is growing.

These data collections contain a wealth of hidden information, that needs to be discovered. Businesses can learn from their transaction data more about the behavior of their customers and therefore can become more efficient by exploiting this knowledge. Science can obtain from observational data (e.g. satellite data, sensor data) new insights on scientific problems. Web usage data can be analyzed and used to optimize information access, but as well to implement novel business models, such as for advertising, on the Web. The task of extracting useful information from large datasets is called **data mining (or data analytics)** and has in the recent years become one of the most dynamically evolving areas in computer science in general.

Data Mining

Most of this data is useless



pictures of drunk people



videos of cats



royal baby, Justin Bieber



SMS to girlfriend

Having increasing amounts of data does not imply having more useful information.

Data Mining Challenge

Transform masses of data into actionable intelligence

- From transaction data to market insights
- From observational data (satellite, sensors) to new scientific hypothesis
- From web usage data to better user interfaces
- ...

Analysis of large data sets to find relationships and to summarize the data in novel ways that are both understandable and useful

The real challenge in data analytics is extracting masses of data into what is usually called insights or actionable insights. Actionable is related to different aspects;

- understandable: the insights need to be interpreted by a human
- Useful: this implies that the insights help to take decision that have practical impact and utility; this also implies that insights that are not « surprising » but just reproduce existing knowledge are also not very useful.

Tackling the Data Mining Challenge

Practical Questions

- Data Access (ownership!)
- Domain knowledge (expertise!)

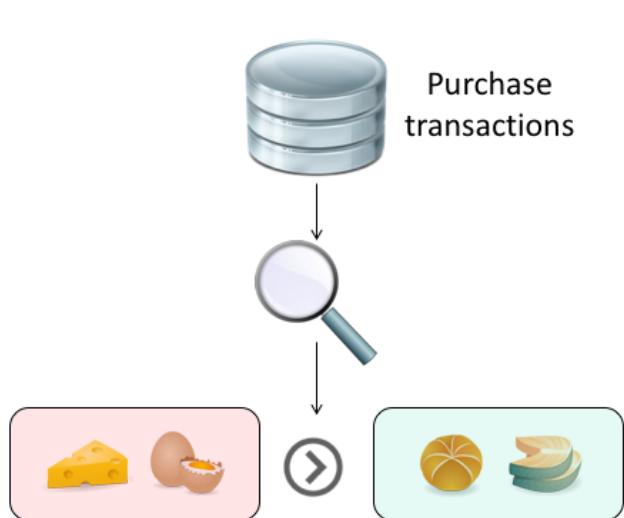
Technical Questions

- Data management (store, index, retrieve): this is referred to as Big Data
- Data mining: algorithmic approaches to produce insights

The challenges to achieve this are manifold, but at least the following three key ingredients are required:

1. The data: of course massive amounts of data exist, but often they are not easily accessible, protected for legal, organizational, economic or political reasons, and spread out in many different systems.
2. The questions: searching for insights into data requires to have at least a general idea of what we are looking for, or what could be an interesting and useful insight. Without having an objective in mind it is hard to find answers.
3. The algorithms: Extracting interesting information out of data, requires efficient and smart algorithms. This is what we will study in the following. Once we master and understand such algorithms, the real challenges will be point 1 and 2.
4. Systems for handling Big Data: this is an area that has made huge progress in the recent years, in particular due to the needs of the big Internet companies. Many of the tools are now available as open source and within the cloud.

Example: Shopping Basket Analysis



- Wal-Mart
- On Fri afternoon 60% of people who buy diapers also buy beer
 - Move beer close to diapers to sell more beers!

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 11

The classical example of a data mining problem is "market basket analysis". Retail stores gather information on what items are purchased by their customers. The expectation is, by finding out what products are frequently purchased together (i.e., are associated with each other), identifying ways to optimize the sales of the products by better targeting certain groups of customers (e.g., by planning the layout of a store or planning advertisements). A well-known example was the discovery that people who buy diapers also frequently buy beers (probably exhausted fathers of small children). Therefore nowadays one finds frequently beer close to diapers in supermarkets, and of course also chips close to beer. Similarly, amazon exploits this type of associations in order to propose to their customers books that are likely to match their interests. This type of problem was the starting point for one of the best known data mining techniques: **association rule mining**.

relate items together

Other Examples

Amazon

Frequently Bought Together



Price for both: \$104.22

Add both to Cart Add both to Wish List

Show availability and shipping details

This item: Energy and the Wealth of Nations: Understanding the Biophysical Economy by Charles A.S. Hall Hardcover \$72.68

Peeking at Peak Oil by Kjell Aleklett Hardcover \$31.54

Analysis of Query Logs (Query Expansion)

- Users that search for “Obama President” often also search for “Obama President Elections”

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 12

The original term of “market basket analysis” does not imply that this problem is limited to analyzing shopping behaviors. The same techniques can and have been applied in many other contexts, including recommender systems, search systems etc.

Classes of Data Mining Problems

Local properties

- Patterns that apply to part of the data
 - e.g., buy diapers → buy beers

Global model

- Descriptive **structure** of the data
 - e.g., 3 types of customer behaviour
- Predictive **function** of the data
 - e.g., $\text{dist}(\text{beer}, \text{diapers})=1 \rightarrow +10\% \text{ beer sales}$

Association rule mining is just one example of a data mining algorithm, out of a wide variety. Data mining algorithms can in general be classified according to the goals they pursue and the type of results they provide.

A basic distinction is made among data mining algorithms that identify global structures of a data set, either in the form of summaries of the data or as globally applicable rules, and data mining algorithms that provide models that apply only locally, i.e., to some subset of the data set, in the form of sparsely occurring patterns or exceptional and unexpected dependencies in the data set.

The example of association rule mining is a typical case of discovering local patterns. The rules obtained are unexpected patterns and typically relate to small parts of the database only.

Data mining algorithms for finding global models of the data are further distinguished into techniques that are used to "simply" describe the data and into techniques that allow to make predictions on data that has not yet been seen. Descriptive modeling techniques provide compact summaries of the data sets, typically by identifying clusters of similar data items. Predictive modeling techniques provide globally applicable rules for the database, typically, allowing to predict some properties of the data, from some other data in the dataset.

global structure:

- summaries of the data
- globally applicable rules

F

local structure (association rule mining)

- sparsely occurring patterns
- exceptional unexpected dependencies

Data Mining Overview

Local properties		Global model	
Patterns		Descriptive model	Predictive model
Association rules	Pattern mining	Clustering	Classification
		Information retrieval	Regression
Exploratory data analysis			
Interactive tools		Visualisation	

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 14

To complete the picture we can include into the field of data mining also exploratory data analysis, a special form of data mining which is used when no clear idea exists, what is being sought for in a given database. It may serve as a preprocessing step for performing more specific data mining tasks.

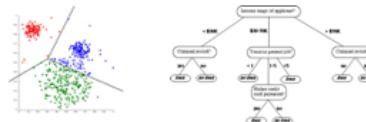
Information retrieval is usually also considered as a special case of data mining, where query patterns are searched for in the database. It can be understood as a clustering technique that distinguishes relevant from non-relevant data items.

why is information
retrieval a descriptive
model

Components of Data Mining Algorithms

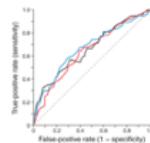
1. Pattern structure/model representation

- What we look for?



2. Scoring function

- How well the model fits the data set?



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 15

Each data mining algorithm can be characterized by four aspects:

- The models or patterns that are used to describe what is searched for in the data set. Typical examples of models are dependencies, clusters and decision trees.
- The scoring functions that are used to determine how well a given data set fits the model. This is comparable to the similarity functions used in information retrieval.
- The method that is applied in order to find data in the data set that scores well with respect to the scoring function. Normally this requires efficient search algorithms that allow to identify those models that fit the data well according to the scoring functions.
- Finally the scalable implementation of the method for large data sets. Here indexing techniques and efficient secondary storage management are typically required. This corresponds to the use of inverted files in information retrieval.

parameter tuning ?

Components of Data Mining Algorithms

3. Optimisation and search

- How to tune the parameters of the model?
[opt]
- How to find data satisfying a pattern? [search]

4. Data management

- How to implement the algorithm for very large data sets?

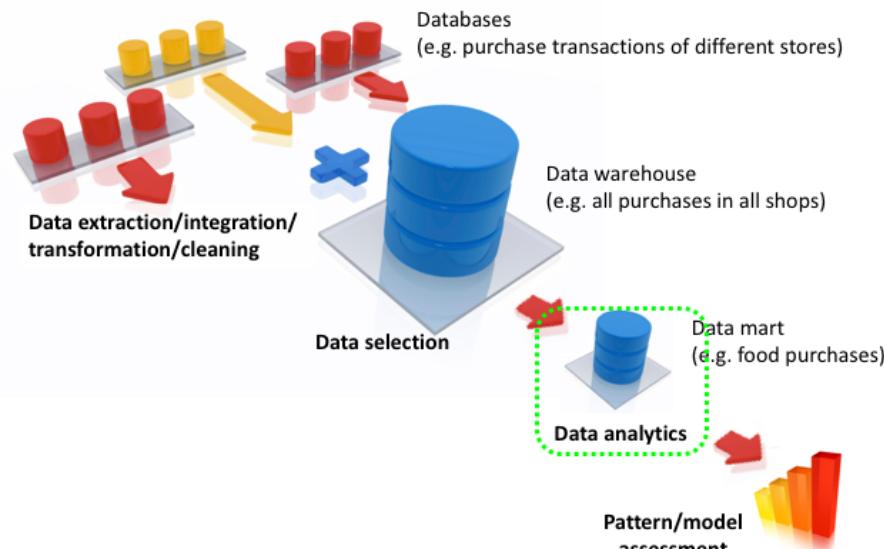


©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 16

In particular the aspects of optimization and data management differentiate data mining from related areas such as statistics and machine learning: data mining algorithms can be understood as statistical or machine learning techniques that scale well to large data sets.

Data Mining System



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 17

Data mining algorithms are part of larger data mining system that support the pre- an post-processing of the data. A data mining systems performs the following typical tasks:

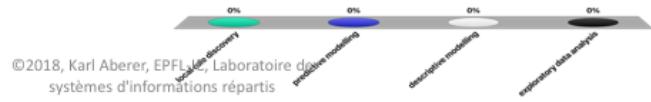
- First, the data needs to be collected from the available data sources. Since **these data sources can be distributed and heterogeneous databases**, recall lecture 1 database integration techniques are applied. The integrated data is kept in so-called data warehouses, databases replicating and consolidating the data from the various data sources. An important concern when integrating the data sources is data cleaning, i.e., removing inconsistent and faulty data as far as possible. The tasks of data integration and data cleaning are supported by so-called data warehousing systems.
- Once the data is consolidated in a data warehouse, subsets of the data can be selected from the data warehouse for performing specific data mining tasks, i.e., tasks targeting a specific question. This task-specific data collections are called data-marts. The data-mart is the database to which the specific data mining algorithm is applied.
- The data mining task is the process of detecting interesting patterns in the data. This is what generally is understood as data mining in the narrow sense. We will introduce in the following examples of the most common techniques that are used to perform this task (e.g. association rule mining).
- Once specific patterns are detected they can be further processed. Further processing may include the evaluation of the "interestingness" of patterns for the specific problem at hand and the implementation of actions to react on the

discovery of a pattern.

Each of the steps described can influence the preceding steps. For example, patterns or outliers detected during data mining may indicate the presence of erroneous data, rather than interesting features in the source databases. This may imply adaptations of the data cleaning process during data integration.

An insurance company wants to find typical, but unknown, causes for specific injuries from earlier cases. This likely is based on ...

- A. local rule discovery
- B. predictive modelling
- C. descriptive modelling
- D. exploratory data analysis



2. ASSOCIATION RULE MINING

1. Pattern Structure

We search association rules of the form

Body \rightarrow Head [support, confidence]

predicate = verb
eg: buy (John, milk)

1. Body: predicate(x , $x \in \{\text{items}\}$) a property of x
2. Head: predicate(x , $x \in \{\text{items}\}$) a property likely to be implied by the Body
3. Support, confidence: measures of the validity of the rule

Example: $\text{buy}(x, \{\text{diapers}\}) \rightarrow \text{buy}(x, \{\text{beer}\}) [0.5\%, 60\%]$

someone who buys diapers also buys beer

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 20

Association rule mining is a technique for discovering unsuspected data dependencies and is one of the best known data mining techniques. The basic idea is to identify from a given database, consisting of item sets (e.g., shopping baskets), whether the occurrence of specific items, implies also the occurrence of other items with a high probability. In principle, the answer to this question could be easily found by an exhaustive exploration of all possible dependencies, which is however prohibitively expensive. Association rule mining thus solves the problem of how to search efficiently for those dependencies.

support: joint
confidence: conditional

Single- and Multi-dimensional Rules

Single-dimensional rules

$\text{buy}(x, \{\text{diapers}\}) \rightarrow \text{buy}(x, \{\text{beer}\}) [0.5\%, 60\%]$

Multi-dimensional rules

$\text{age}(x, \{19-25\}) \wedge \text{buy}(x, \{\text{chips}\}) \rightarrow \text{buy}(x, \{\text{coke}\}) [10\%, 75\%]$

As shown in the previous example, we can have two kinds of association rules. Those that associate data for a single predicate (such as buys), which are called single-dimensional rules, and those that use several different predicates (such as buys and age), which are called multi-dimensional rules.

From Multi- to Single-dimensional Rules

Use predicate/value pairs as items

$$\text{age}(x, \{19-25\}) \wedge \text{buy}(x, \{\text{chips}\}) \rightarrow \text{buy}(x, \{\text{coke}\})$$

$$\text{customer}(x, \{\text{age}=19-25\}) \wedge \text{customer}(x, \{\text{buy}=\text{chips}\}) \rightarrow \text{customer}(x, \{\text{buy}=\text{coke}\})$$

Simplified notation of single-dimensional rules

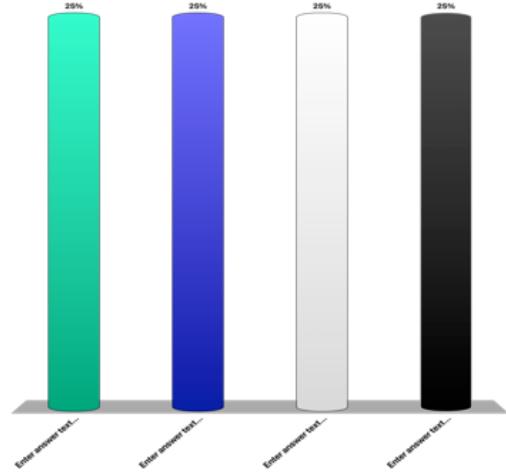
$$\{\text{diapers}\} \rightarrow \{\text{coke}\}$$
$$\{\text{age}=19-25, \text{buy}=\text{chips}\} \rightarrow \{\text{buy}=\text{coke}\}$$

However, it is straightforward to transform multi-dimensional association rules into single-dimensional rules, by considering different predicates applied to the same items as different items. Therefore in the following we will only consider single-dimensional association rules. As a result we will also use a simplified notation for the rules, where instead of using predicates we just distinguish different items.

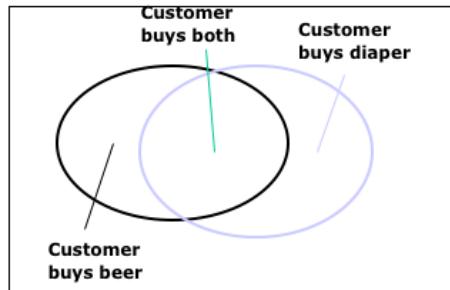
todo

Let's assume that the transactions are stored in a relation $T(x, A1, \dots, A5)$, where x is the customer and each attribute $A1, \dots, A5$ can have 3 different values. How many different items exist after reduction to a single dimension?

- A. 5
- B. 243
- C. 125
- D. 15



2. Scoring Function



Transaction ID	Items Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, eggs, apple

Support: probability that body and head occur in transaction
 $p(\{\text{body}, \text{head}\})$

Confidence: probability that if body occurs, also head occurs
 $p(\{\text{head}\} \mid \{\text{body}\})$

This example illustrates the basic concepts used in association rule mining.

Transactions consist of a transaction identifier and an *itemset*. The itemset is the set of *items* that occur jointly in a transaction (e.g., the items bought). Now we can define the measures used to evaluate the quality of a rule.

Support is the number of transactions in which the association rule holds, i.e., in which all items of the rule occur (e.g., both beer and diaper). If this number is too small, probably the rule is not relevant. *Confidence* is the probability that in case the body of the rule (the condition) is satisfied also the head of the rule (the conclusion) is satisfied. This indicates to which degree the rule is satisfied, in those cases where it is applicable.

support: joint

confidence: conditional

itemset: subset of all items. per row

Support and Confidence

Transaction ID	Items Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, eggs, apple

Rule1: {beer} \rightarrow {diaper}

$$p(\{beer, diaper\}) = 2/4$$

$$p(\{diaper\} \mid \{beer\}) = 2/3$$

{beer} \rightarrow {diaper}[50%, 66%]

Rule2: {diaper} \rightarrow {beer}

$$p(\{diaper, beer\}) = 2/4$$

$$p(\{beer\} \mid \{diaper\}) = 2/2$$

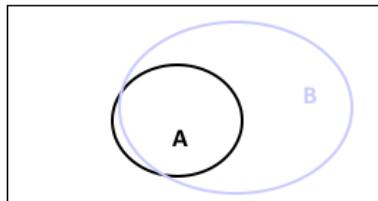
{diaper} \rightarrow {beer}[50%, 100%]

Here we compute support and confidence for two different rules.

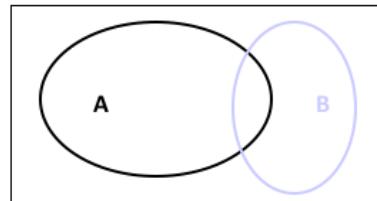
support: joint
confidence: conditional
[support, confidence]

Support and Confidence

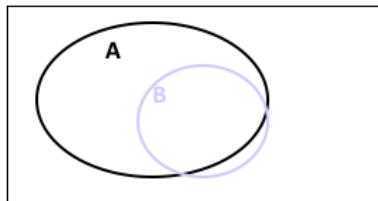
Let's assume $p(\{A, B\})$ is above threshold (high support)



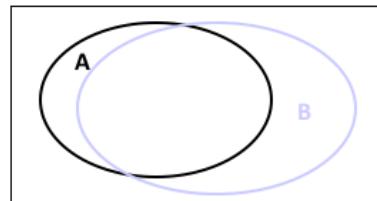
$\text{conf}(A \rightarrow B)$: high
 $\text{conf}(B \rightarrow A)$: low



$\text{conf}(A \rightarrow B)$: low
 $\text{conf}(B \rightarrow A)$: low



$\text{conf}(A \rightarrow B)$: low
 $\text{conf}(B \rightarrow A)$: high



$\text{conf}(A \rightarrow B)$: high
 $\text{conf}(B \rightarrow A)$: high

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 26

Here we illustrate the meaning and importance of the "direction" of an association rule. We assume that in all cases the intersection areas (i.e., the support) are above the required threshold. Then four combinations for the confidence are possible. Thus association rules not only express a high probability of co-occurrence of items, such as in the last case, where confidence is high in both directions, but also conditional dependencies among the occurrences of items (or inclusion relationships).

support: joint
confidence: conditional

confidence > support = body positively implies head
confidence < support = body negatively implies head

Definition of Association Rules

Terminology and Notation

- Set of all items I , subset of I is called **itemset**
- **Transaction** (tid, T), $T \subseteq I$ itemset, transaction identifier tid
- Set of all transactions D (**database**), Transaction $T \in D$

Association Rules $A \rightarrow B [s, c]$

- A, B itemsets ($A, B \subseteq I$)
- $A \cap B$ empty
- $s = p(A \cup B) = \text{count}(A \cup B) / |D|$ (support)
- $c = p(B | A) = s(A \cup B) / s(A)$ (confidence)

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 27

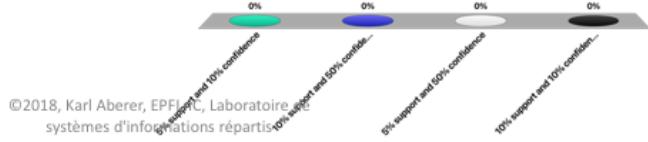
Here we summarize the basic concepts that constitute the structure of the association rule patterns and the measures for their evaluation that are used in association rule mining.

The support is computed as the number of occurrences of A and B together in a transaction, divided by the total number of transactions. This corresponds to the probability that a transaction contains $A \cup B$.

The confidence is computed as the support of $A \cup B$ divided by the support of A , that is, the frequency of the elements in the union of A and B in a transaction divided by the frequency of A . This corresponds to the conditional probability that a transaction having A also contains B

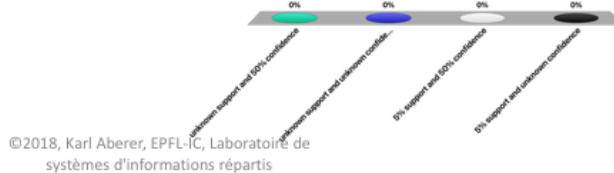
10 itemsets out of 100 contain item A, of which 5 also contain B. The rule A → B has:

- A. 5% support and 10% confidence
- B. 10% support and 50% confidence
- C. 5% support and 50% confidence
- D. 10% support and 10% confidence



10 itemsets out of 100 contain item A, of which 5 also contain B. The rule B → A has:

- A. unknown support and 50% confidence
- B. unknown support and unknown confidence
- C. 5% support and 50% confidence
- D. 5% support and unknown confidence



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Association Rule Mining: Problem

Problem

Given a database D of transactions (tid, T)

Find

all rules $A \rightarrow B [s, c]$

such that $s > s_{\min}$ (high support)

and $c > c_{\min}$ (high confidence)

The problem of mining association rules is now defined as follows:

Given a database of transactions, find all rules that correlate the presence of one set of items with that of another set of items where the support and the confidence are above a given threshold. In other words, find all rules that have high support and confidence.

support: joint
confidence: conditional

Association Rule Mining Approach

Two step approach:

1. Find **frequent** itemsets

$$J \subseteq I \text{ such that } p(J) > s_{min}$$

2. Select **pertinent** rules

$$A \rightarrow B \text{ such that } A \cup B = J$$

and

$$p(B|A) > c_{min}$$

We will approach the problem in two steps, by first considering only the problem of finding itemsets that satisfy the condition on having a high support, which is a necessary condition for a rule to be an association rule, and then extracting from those itemsets the rules that have a high confidence.

Step 1: Find frequent itemsets with high support

Step 2: Then find frequent itemsets with high confidence

Having high support does not mean high confidence

Frequent Itemsets

Transaction ID	Items Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, eggs, apple

Frequent Itemset	Support
{beer}	0.75
{milk}	0.75
{diaper}	0.5
{beer,milk}	0.5
{beer, diaper}	0.5
{milk, diaper}	0.25

1. $A \rightarrow B$ can be an association rule, only if $A \cup B$ is a frequent itemset
2. Any subset of a frequent itemset is also a frequent itemset (**Apriori property**)

$$p(J) > s_{min} \rightarrow p(J' \subseteq J) > s_{min}$$
3. Find frequent itemsets with increasing cardinality, from 1 to k, to reduce the search space

A necessary condition for finding an association rule of the form $A \rightarrow B$ is that it has a sufficiently high support. Therefore, for finding such rules, we can first try to find itemsets within the transactions that occur sufficiently frequent. These are called *frequent itemsets*. Second we can observe that any subset of a frequent itemset is necessarily also a frequent itemset. This is called the *apriori property*. The a priori property is a central idea in the algorithm we will introduce that will be used for multiple purposes. A first use of this property is the observation that we can search for frequent itemsets by searching them by increasing cardinality: once frequent itemsets of lower cardinality are found, only itemsets of larger cardinality need to be considered that contain one of the frequent itemsets already found. This allows us to dramatically reduce the search space.

support: joint
confidence: conditional

if beer AND milk is frequent, then individually, beer and milk must be frequent

Exploiting the Apriori Property

(k-1) subsets must all be frequent

If we know the frequent (k-1)-itemsets, which is a candidate frequent k-itemset X?

1. Any (k-1)-itemset that is subset of X must be frequent
2. Any ordered (k-1)-itemset that is subset of X must be frequent
3. *In particular, two ordered (k-1)-itemsets that are subsets of X and differ only in their last position must be frequent*

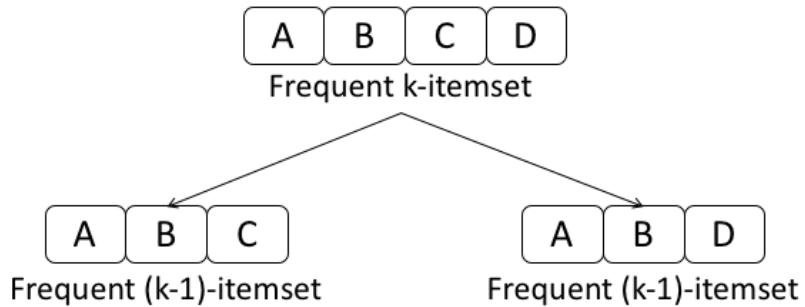
Assume that we know frequent itemsets of size k-1 and want to construct a itemset of size k (a k-itemset). What properties does such a k-itemset has to satisfy?

We can definitely say, that any subset of size k-1 must be frequent. We can also order those subsets. We can that even say that two order subsets that **differ only in the last position** must be frequent. This gives is now a possibility to construct k-itemsets systematically.

todo: why must differ in last position ?

[https://
stackoverflow.com/
questions/20404833/
lexicographical-order-
in-apriori-algorithm](https://stackoverflow.com/questions/20404833/lexicographical-order-in-apriori-algorithm)

Exploiting the Apriori Property: Candidates



The union of two $(k-1)$ -itemsets that differs only by one item is a **candidate** frequent k -itemset

By inverting the argument, we can see this also as a way to construct frequent k -itemsets. We take two $(k-1)$ item sets which differ only by one item and take their union. This step is called the join step and is used to construct POTENTIAL frequent k -itemsets (called candidate k -itemset)

Example

Transaction ID	Items Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, eggs, apple

Frequent Itemset	Support
{beer}	0.75
{milk}	0.75
{diaper}	0.5
{beer,milk}	0.5
{beer, diaper}	0.5
{milk, diaper}	0.25

{beer} and {milk} differ by one item, thus {beer,milk} is a candidate 2-itemset

{beer, milk} and {beer, diaper} differ by one item, thus {beer, diaper, milk} is a candidate 3-itemset

Note that the candidate itemsets generated by the join step are not necessarily frequent itemsets, as the example illustrates. The condition we use is necessary, but not sufficient.

sufficient is stronger
than necessary

Exploiting the Apriori Property: JOIN step

Given the frequent $(k-1)$ -itemsets, L_{k-1} , we can construct a candidate set C_k by joining two $(k-1)$ -itemsets that differ by exactly 1 item in the last position

Algorithm (**JOIN**)

1. Sort the itemsets in L_{k-1} sort lexicographically
2. Find all pairs with the same first $k-2$ items, but different $k-1^{\text{th}}$ item
3. Join the two itemsets in all pairs and add to C_k

The construction of potential k -itemsets, so-called candidates, from $(k-1)$ -itemsets, can be organized in a way that minimizes the number of candidates being generated. By sorting the items only items that are the highest in the order are the ones being combined from different $(k-1)$ -itemsets. For the remaining $k-2$ items we have just to consider just those pairs of $(k-1)$ -itemsets that coincide on all of them.

We only combine the itemsets that have the first $k-2$ identical elements in order to avoid the generation of duplicate candidates. For example if we have

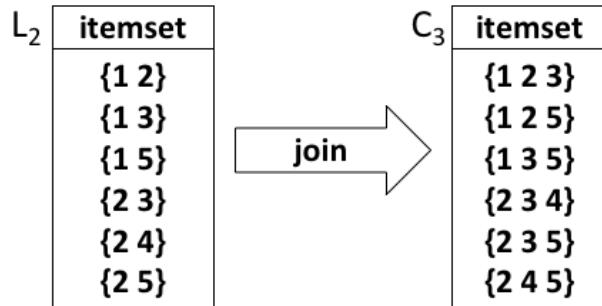
ABC

ACD

BCD

If we combine ACD and BCD (which also differ by one item) we would get ABCD. But ABCD have been already generated by combining ABC and ABD.

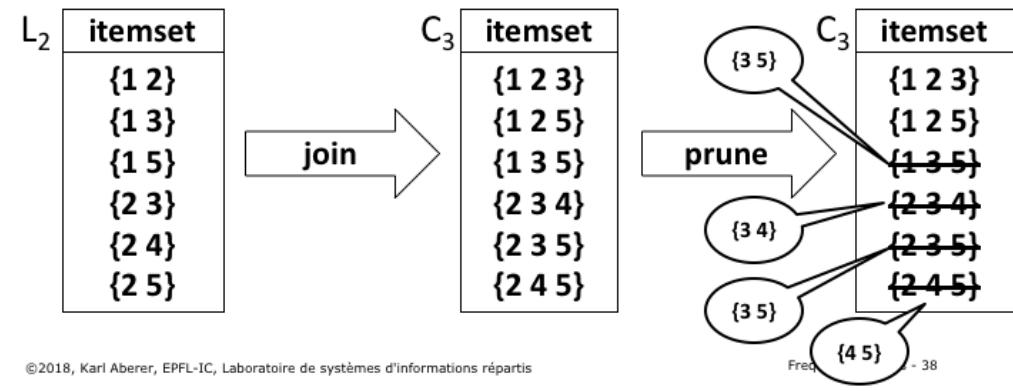
JOIN step: Example



join step does not
delete anything

Exploiting the Apriori Property: PRUNE Step

A k-itemset in the candidate set C_k might still contain $(k-1)$ -itemsets that are not frequent $(k-1)$ -itemsets
 Eliminate them (**PRUNE**)



The k-itemsets constructed in the join step are not necessarily frequent k-itemsets. One possible reason is that they contain some subsets of items which are not frequent. These are next eliminated in a prune step, by checking if every $(k-1)$ itemset of the candidate itemset is indeed a frequent $(k-1)$ itemset.

$\{2,4,5\} = \{2,4\}, \{2,5\}, \{4,5\}$

use candidate set to prune. pruning here is not based on support
 use itemset to merge

Final Step

For the remaining k-itemset in the candidate set C_k eliminate those that are not frequent by counting how often they occur in the database

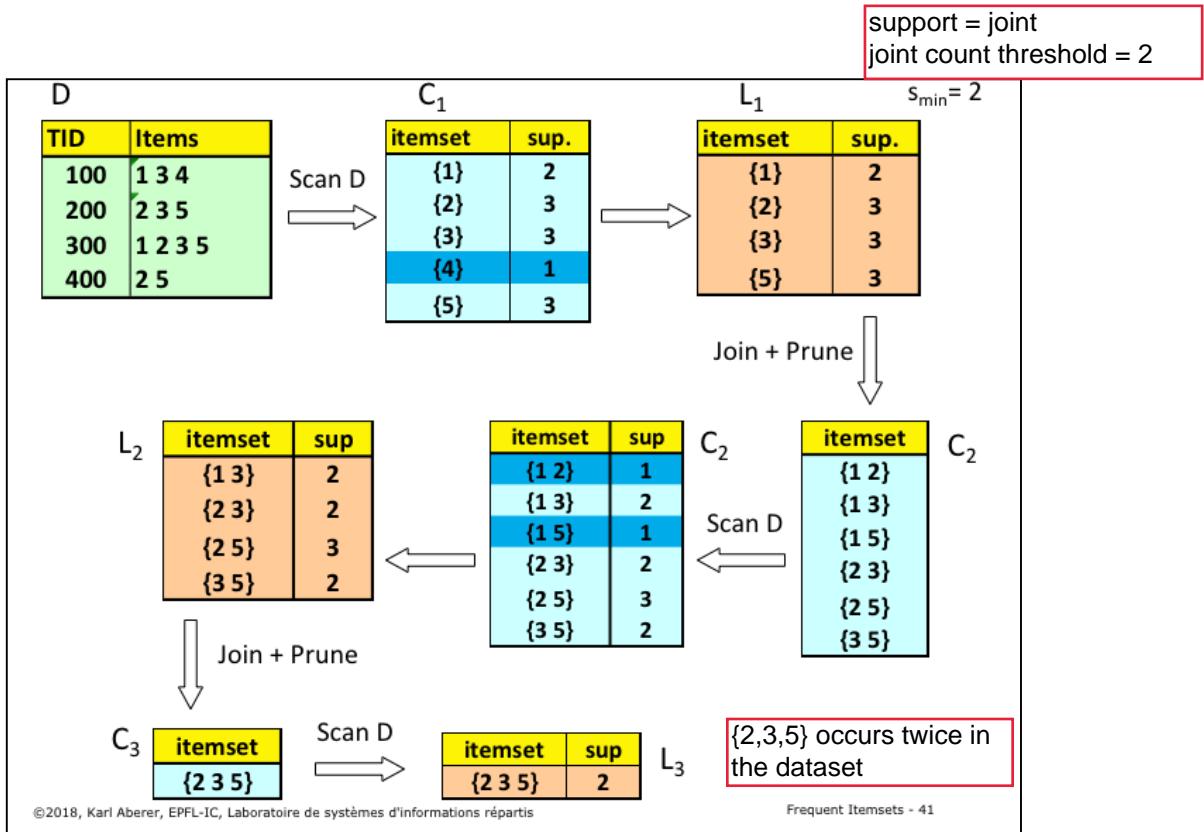
- The final step is the most expensive (full access to database D)
- Advantage: Performed only for a smaller number of candidate itemsets, reduced by **JOIN** and **PRUNE**

After this step is completed, it needs to be checked which of the candidate itemsets constructed so far are indeed frequent. For this purpose the frequency of these itemsets needs to be determined by accessing the database. The important aspect of the apriori algorithm is that this last step is the most expensive one, since it requires access to the complete database, but needs to be performed for a smaller number of itemsets, since many possibilities have been eliminated in the join and prune step.

Apriori Algorithm

```
k := 1, Lk := { J ⊆ I : |J| = 1 ∧ p(J) > smin }           //frequent items
while Lk ≠ ∅
    C'_{k+1} := JOIN(Lk)           // join itemsets in Lk that differ by one element
    C_{k+1} := PRUNE(C'_{k+1})        // remove non-frequent itemsets
    for all (id,T) ∈ D             // compute the frequency of candidate
        for all J ∈ C_{k+1}, J ⊆ T
            count(J)++
        end for
    end for
    L_{k+1} := { J ⊆ C_{k+1} : p(J) > smin }           // add candidate frequent itemsets
    k := k+1
end while
return ∪k Lk
```

This is the summary of the complete apriori algorithm.



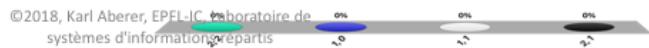
Here we give a complete example for the apriori algorithm. Notice, in this example, how the scan steps (when determining the frequency with respect to the database) eliminate certain itemsets. Pruning does not lead to any elimination of itemsets in this example. The algorithm built 9+3 (frequent + non frequent) = 12 itemsets out of $2^5 = 32$ total possible itemsets

ONLY FILTER OUT ITEMSETS BASED ON SUPPORT AFTER JOIN + PRUNE STEP

Given the frequent 2-itemsets $\{1,2\}$, $\{1,4\}$, $\{2,3\}$ and $\{3,4\}$, how many 3-itemsets are generated and how many are pruned?

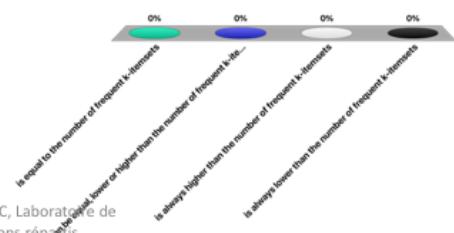
- A. 2, 2
- B. 1, 0
- C. 1, 1
- D. 2, 1

©2018, Karl Aberer, EPFL-IC Laboratoire de
systèmes d'informations répartis



After the join step, the number of k+1-itemsets ...

- A. is equal to the number of frequent k-itemsets
- B. can be equal, lower or higher than the number of frequent k-itemsets
- C. is always higher than the number of frequent k-itemsets
- D. is always lower than the number of frequent k-itemsets



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Select pertinent rules

```
R := ∅                                // initial set of rules
L := Apriori(D)                         // set of frequent itemsets
for all J ∈ L
    for all A ⊆ J, A ≠ ∅
        r := A → J \ A                  // create candidate rule
        if c(A → J \ A) = s(J)/s(A) > cmin
            R := R ∪ r
        end if
    end for
end for
```

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 44

Once the frequent itemsets are found with Apriori, the derivation of pertinent association rules is straightforward.

One checks for every frequent itemset whether there exists a subset A that can occur as the body of a rule. For doing that, the support count, i.e., the frequency of the itemset in the database, which was obtained during the execution of the Apriori algorithm, is used to compute the confidence as a conditional probability. Note that also L\A is a frequent itemset, and therefore the support count is available for that set from the Apriori algorithm.

Note that $c(A \rightarrow J \setminus A) = s(J \setminus A \cup A)/s(A) = s(J)/s(A)$

what is this ?

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

$c_{min} = 0.75$

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2
{2 3 5}	2

$J=\{1,3\}$

$A=\{1\}, J \setminus A = \{3\}$ $\{1\} \rightarrow \{3\}$ $c = \text{sup}(\{1,3\})/\text{sup}(\{1\}) = 1$ OK
 $A=\{3\}, J \setminus A = \{1\}$ $\{3\} \rightarrow \{1\}$ $c = \text{sup}(\{1,3\})/\text{sup}(\{3\}) = 0.66$ KO

$J=\{2,3,5\}$

$A=\{3,5\}, J \setminus A = \{2\}$ $\{3,5\} \rightarrow \{2\}$ $c = \text{sup}(\{2,3,5\})/\text{sup}(\{3,5\}) = 1$ OK
 $A=\{2\}, J \setminus A = \{3,5\}$ $\{2\} \rightarrow \{3,5\}$ $c = \text{sup}(\{2,3,5\})/\text{sup}(\{2\}) = 0.66$ KO

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 45

Notice, in this example, of how the scan steps (when determining the frequency with respect to the database) eliminate certain itemsets. Pruning does not lead to any elimination of itemsets in this example. The algorithm built $9+3$ (frequent + non frequent) = 12 itemsets out of $2^5 = 32$ total possible itemsets

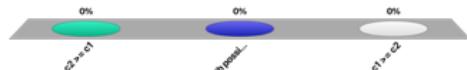
support: joint
confidence: conditional

todo: no need scan for
 $3 \rightarrow 2,5$?

If rule $\{A, B\} \rightarrow \{C\}$ has confidence c_1 and rule $\{A\} \rightarrow \{C\}$ has confidence c_2 , then ...

- A. $c_2 \geq c_1$
- B. $c_1 > c_2$ and $c_2 > c_1$
are both possible
- C. $c_1 \geq c_2$

apply bayes:
 $k \times c_2 = c_1$



©2018, Karl Aberer, EPFL-IC, Laboratoire de
systèmes d'informations répartis

A,B,C
A,C

$$\begin{aligned} P(ABC) / P(AB) &= 1 \\ P(AC) / P(A) &= 0.5 \end{aligned}$$

Interesting Association Rules

Not all high-confidence rules are interesting

	<i>Coffee</i>	$\overline{\text{Coffee}}$	
<i>Tea</i>	150	50	200
$\overline{\text{Tea}}$	650	150	800
	800	200	1000

{Tea} → {Coffee} has support 0.15 and confidence 0.75

- But 80% of people drink coffee anyway
- Drinking tea decreases the probability to drink coffee!

$$p(\text{coffee}) = 0.8$$

$$p(\text{coffee} \mid \text{tea}) = 0.75$$

©2010, Ralf Küller, LFR-LC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 47

High confidence and support to not necessarily imply that a rule is interesting or useful. We illustrate this by the following example. The matrix indicates the number of items that like / do not like coffee or tea. If we consider the rule {Tea} → {coffee} we will find that it has support 0.15 and confidence 0.75, which we may consider as very high. So at the first glance this looks like a good rule. Looking more carefully we will realize that actually 80% of the people drink coffee. In view of this, the rule just holds, because people in general drink a lot of coffee. Even worse, considering that 80% of people drink coffee, among the people drinking tea the propensity to drink coffee is less pronounced, only 75%. So drinking tea has a negative impact on drinking coffee which is the opposite of what the rule suggests. This motivates the need for other measures to evaluate whether a rule is interesting.

support: joint

confidence: conditional

Alternative Measures of Interest

Added Value (A, B itemsets)

$$AV(A \rightarrow B) = confidence(A \rightarrow B) - support(B)$$

Interesting rules are those with high positive or negative interest values (usually above 0.5)

Alternative: $Lift(A \rightarrow B) = \frac{confidence(A \rightarrow B)}{support(B)}$

Example:

- $Lift(\{\text{Tea}\} \rightarrow \{\text{Coffee}\}) = 0.75 / 0.8 = 0.9375$
- $AV(\{\text{Tea}\} \rightarrow \{\text{Coffee}\}) = 0.75 - 0.8 = -0.05$

In order to account for the effect of rule heads with high support, its value can be considered as a correction of the confidence measure. There exist two variants of that approach, either taking the difference or the ratio. A positive value of added value indicates that X and Y are related, a negative indicates that A prevents B from occurring.

Only if the probability of finding item B when item A has been found is greater than the probability of finding item B at all can we say that A implies B.

Lift examples

0.75 / 0.8 = 0.9375, below 1, thus tea "negatively" affects coffee
if 0.75 / 0.6 = 1.25, above 1, thus tea "positively" affects coffee

Added Value is more obvious, since it is subtraction,

negatively affect = negative AV
positively affect = positive AV

Quantitative Attributes

Transforming quantitative (numeric ordered values) into categorical ones

- *Static discretisation* into predefined bins
- *Dynamic discretisation* based on the distribution of the data

The rules depends on the chosen discretisation!!

(1) $\text{age}\{x, [18,19]\} \wedge \text{live}\{x, \text{Lausanne}\} \rightarrow \text{work}\{x, \text{student}\}$

(2) $\text{age}\{x, [20,21]\} \wedge \text{live}\{x, \text{Lausanne}\} \rightarrow \text{work}\{x, \text{student}\}$

vs.

(1) $\text{age}\{x, [18,21]\} \wedge \text{live}\{x, \text{Lausanne}\} \rightarrow \text{work}\{x, \text{student}\}$

previously was coffee -> tea, now is numbers -> something

For quantitative attributes the search for association rule is more complex. A simple approach is to statically or dynamically discretize quantitative attributes into categorical attributes.

However, the rules that can be found depend on the discretization chosen. It may happen that the bins are for example too fine-grained, and a rule that could be expressed in a compact form at a coarser granularity is split into multiple rules.

For example: if age is discretized into steps of 2 years we would probably find rules

$\text{Age}(X, 18..19) \text{ and } \text{lives}(X, \text{Lausanne}) \rightarrow \text{profession}(X, \text{student})$

$\text{Age}(X, 20..21) \text{ and } \text{lives}(X, \text{Lausanne}) \rightarrow \text{profession}(X, \text{student})$

This could be also expressed as a rule

$\text{Age}(X, 18..21) \text{ and } \text{lives}(X, \text{Lausanne}) \rightarrow \text{profession}(X, \text{student})$

which is more compact but requires a different discretization.

Improving Apriori for Large Datasets

1. Transaction reduction

- A transaction that does not contain any frequent k-itemset is useless in subsequent scans

2. Sampling

- Mining on a sampled subset of DB, with a lower support

3. Partitioning (SON algorithm)

- Any itemset that is potentially frequent in a DB must be frequent in at least one of the partitions of the DB

Though the basic Apriori algorithm is designed to work efficiently for large datasets, there exist a number of possible improvements:

- Transactions in the database that turn out to contain no frequent k-itemsets can be omitted in subsequent database scans.
- The sampling method selects samples from the database and searches for frequent itemsets in the sampled database using a correspondingly lower threshold for the support.
- One can try to identify first frequent itemsets in partitions of the database, that fits in memory. This method is based on the assumption that if an itemset is not frequent in one of the partitions at least (local frequent itemset) then it will also not be frequent in the whole database.

reasons for 3 points above

1. todo
2. need to use lower threshold in case a frequent itemset becomes non frequent in the subset
3. if an itemset is not frequent in at least one of the subsets then it is definitely not frequent

Sampling

Approach

1. Randomly sample transactions with probability p
2. Detect frequent itemsets with support p^* 's
3. Eliminate **false positives** by counting frequent itemsets on complete data after discovery

might accidentally sample too much of the same item resulting in false positive. so must verify on the full database

Refinements

- If we assume that the m transactions are randomly sorted, we can just choose the first p^*m ones
- False negatives can be reduced by choosing a lower threshold, e.g. 0.9 p^* 's

because if p^* 's is too high we might WRONGLY REJECT a frequent itemset (false negative)

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent itemsets

Random sampling has to consider two issues: false positives and false negatives. False positives can be dealt with by verifying the candidate itemsets on the complete transaction set. For false negative no such possibility exists. By decreasing the support threshold, the number of false negatives can be reduced, at the cost of testing more candidate itemsets when scanning the complete dataset.

A further optimization can be achieved, when we know that the transactions occur in random order. Then sampling can be replaced by simple using the first p^*m elements of the database.

Partitioning

Approach

1. Divide transactions in $1/p$ partitions and repeatedly read partitions into main memory
2. Detect in-memory algorithm to find all frequent itemsets with support threshold p^* s
3. An itemset becomes a candidate if it is found to be frequent in **at least** one partition
4. On a **second pass**, count all the candidate itemsets and determine which are frequent in the entire set of transactions

With partitioning we are not sampling the dataset, but processing the complete transaction set piece by piece. If an itemset is frequent in one partition (with a correspondingly adapted threshold), then it becomes a candidate. At this stage it is not sure that it is also frequent for the whole transaction set. Therefore in a second pass itemsets that are frequent are determined.

Partitioning – Why it works

Key “monotonicity” idea

- an itemset cannot be frequent in the entire set of transactions unless it is frequent in at least one subset
- Proof: If for all partitions support is below p^* s, the total support is less than $(1/p) p^*s = s!$

The second pass is needed, since the condition of being frequent in at least one partition is necessary, but not sufficient.

Partitioning – Distributed Version

Partitioning lends itself to distributed data mining

Transactions distributed among many nodes

- Compute frequent itemsets at each node
- Distribute candidates to all nodes
- Accumulate the counts of all candidates

MapReduce implementation!

Partitioning is an embarrassingly parallel algorithm, and can therefore be naturally executed in a distributed environment, e.g. using Map-Reduce

A false negative in sampling can only occur for itemset with support smaller than ...

- A. the threshold s
- B. p^*s
- C. p^*m
- D. None of the above



false negative can occur regardless.
eg only 1 itemset (that is actually frequent) in the subset

FP Growth

todo: what does candidate here mean

Frequent itemset discovery without candidate itemset generation

Proceeds in 2 steps:

1. Build a data structure, called the FP-tree
 - Requires 2 passes over the dataset
2. Extract frequent itemsets directly from the FP-tree

Though Apriori is as the original method the most popular rule mining algorithm, and also frequently used, other algorithms have been devised that attempt to provide better performance (under certain circumstances). FP Growth is one example of such an algorithm that is mainly aiming at main memory implementations (and thus less suitable for distributed implementation).

FP-Tree Data Structure

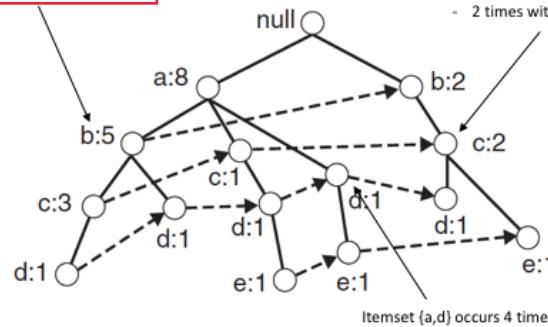
itemset abcd occurs 1 time
itemset abc occurs 3 times

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Itemset {a,b} occurs 5 times

Itemset {a,b} occurs 5 times:
 - 3 times with a
 - 2 times without a

item b occurs 7 times
itemset {a,b} occurs 5 times



Itemset {a,d} occurs 4 times

- Items are sorted
- Nodes correspond to single items
- Counters indicate frequency of itemset from root to node
- Different nodes for the same item are connected by a linked list

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

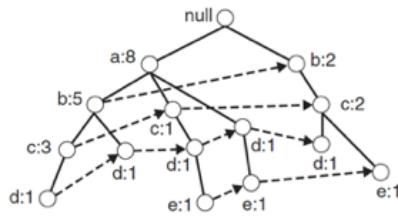
Frequent Itemsets - 57

FP Growth is based on the construction of a data structure, called FP-Tree. The core structure of and FP-Tree as similar to a trie. In order to construct the FP-Tree, first items in the itemsets are sorted. The nodes in the tree correspond to items, and paths from the root correspond to itemsets. Itemsets sharing the same prefix, share also the same path prefix in the tree. Counters at the nodes indicate how frequent the itemset corresponding to the path from the root to the node is. If we consider the left most path, we see that item a occurs 8 times, itemset ab 5 times, abc 3 times, and abcd once. Since the same item can occur in the tree in multiple places, these different occurrences are connected as linked list. For itemsets that occur in different paths (e.g. bc, which occurs together with a and without a), the frequencies can be computed by following the linked lists of the last item in the itemset and testing whether the other items are present in the path.

FP-Tree: Item Sorting

Transaction Data Set

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



Item frequencies
 a: 8
 b: 7
 c: 6
 d: 5
 e: 3

FP-Tree Size: The FP-Tree is more compact the more common prefixes the itemsets share

- Sorting items by decreasing support increases this probability

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 58

Sorting items in itemsets by decreasing support is an important heuristics to keep the FP-Tree compact, as more frequent items tend to be on the top of the tree which increases the likelihood that different itemsets share a common prefix. In the example we had sorted the items this way. If it is not the case they are resorted according to their frequencies.

the letters of each itemset must be sorted according to their frequencies.
 it just so happens that they are also in alphabetical order here

FP-Tree Construction

Requires 2 passes

Pass 1: Compute support for each item

- Pass over the transaction set
- Sort items in order of decreasing support

Pass 2: Construct the FP-Tree data structure

- Tree is expanded one itemset at a time

To construct the FP-Tree one proceeds in two passes (not to confuse with the two steps of the overall FP-Growth algorithm mentioned in the beginning). In a first pass the transactions are scanned to compute the support for each single item. The items are then sorted in decreasing order. This order is being used to sort itemsets as required by the specification of the FP-Tree.

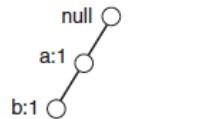
In the second pass the FP-Tree structure itself is constructed by adding one itemset at a time.

sort ITEMS in the ITEMSET according to support of ITEM

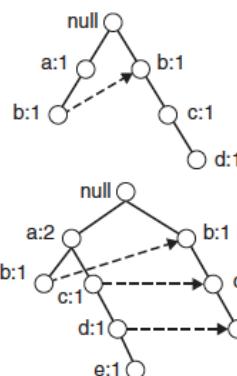
Step 1: FP-Tree Construction

Transaction Data Set

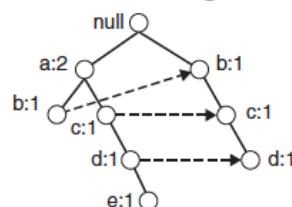
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



Adding itemset 1



Adding itemset 2



Adding itemset 3

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 60

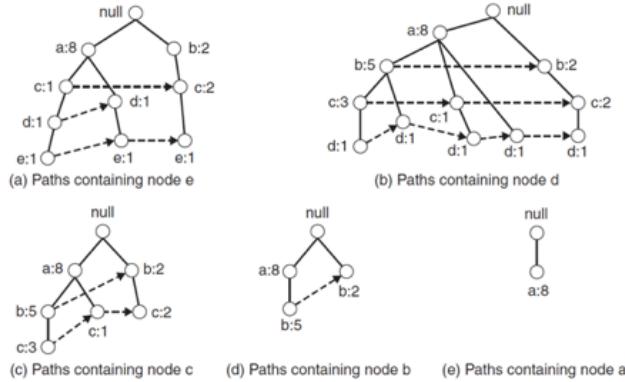
Constructing the FP-Tree proceeds in a way analogous to constructing a trie structure. The items in the itemset are processed in their order. If for the current item there exists already a node, no new node is created. If it does not exist, a new branch is created for the items. The number of occurrences is updated by 1 at each node that is traversed (since each subset of the itemset is also an itemset). Finally links are introduced between nodes with the same labels.

We can traverse the links to count the occurrence for that item

Step 2: Frequent Itemset Extraction

Bottom-up approach

- For each item extract the tree with paths ending in the item



In the second step frequent itemsets are extracted. For each item first a subtree is extracted from the full FP-Tree that contains only the paths with the item. For extracting those subtrees the linked lists are helpful. One needs to traverse just the corresponding list and retain the paths from the traversed nodes to the root.

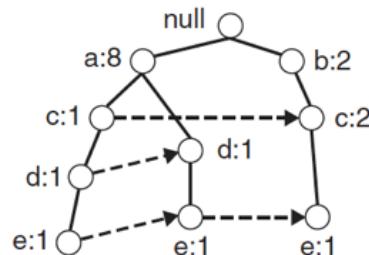
The figures illustrates the five resulting subtrees.

Divide and Conquer Strategy

Start with item with lowest support (e.g. item e)

Check whether the item has sufficient support

- Add up counts along the list of the item
- E.g. item e has support 3



number of external nodes = highest possible support of all itemsets containing e

to get external nodes, find all subtree that end in that node

eg: linked list connecting e

©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 62

Then start processing for the item with the lowest support. First we verify that the item has sufficient support. This we can simply do by following the list and adding up all the counts found. In the example, item e would this have support 3. If the support is above threshold processing proceeds, otherwise we know that no itemset containing e has sufficient support.

- item with lowest support at the bottom
- if item < threshold, then no itemset that contains e will be a frequent itemset

Divide and Conquer Strategy

If sufficient support, check for itemsets ending in the item

- Example: if item *e* is frequent check for itemsets ending in *de*, *ce*, *be*, *ae*

Derive **conditional FP-Trees**

1. Update support counts to itemsets containing the item
2. Remove the nodes of the item
3. Remove nodes with insufficient support count

If we find that the item at hand (e.g. item *e*) has sufficient support, then we check next whether the itemsets consisting of two items and ending in the item have also sufficient support. For doing so we derive now a conditional FP-Tree from the tree that we have constructed before for the item. The conditional tree is constructed in three steps:

- First the support counts in the tree are updated such that only the number of itemsets containing the current item is considered. This can be simply performed traversing the paths from the bottom to the root.
- Then the nodes corresponding to the current item are removed
- Finally all nodes that have an insufficient support count are also removed from the conditional FP Tree why ?

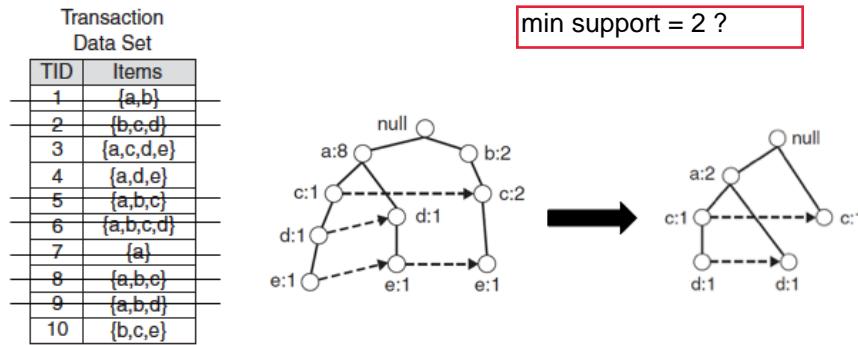
- update count of tree containing *e*
 - remove *e*
 - remove all nodes that have insufficient support

Conditional FP-Tree

conditional FP tree - support count of items containing e

The FP-Tree that would be built if we only consider transactions containing a particular itemset (e.g. e)

- Removing that itemset from the tree and dropping items that are not frequent



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Frequent Itemsets - 64

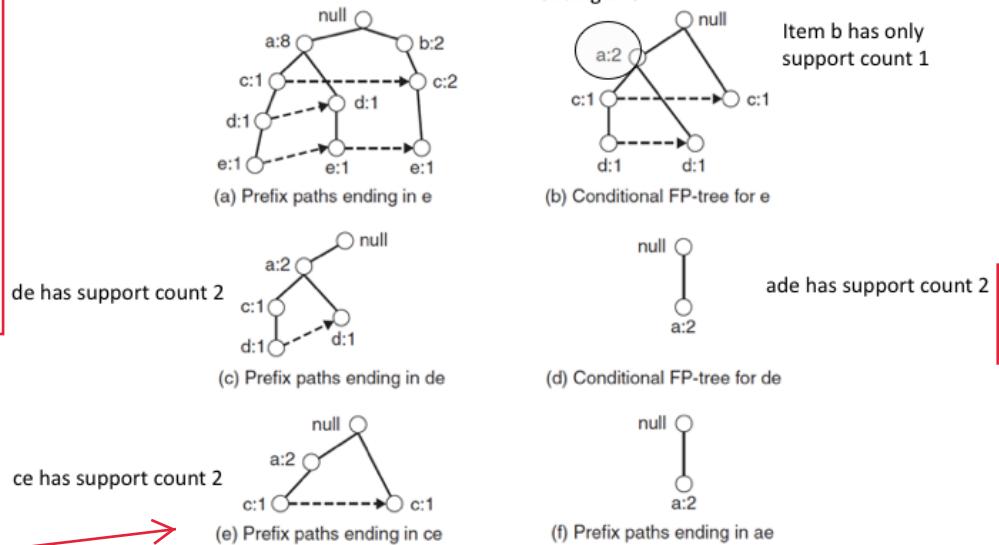
Another way to understand the construction of the conditional FP Tree is that it is the FP Tree that would be constructed when we only consider the transactions that consider the current item (or itemset) and then remove e and any non-frequent item from those itemsets. In the example this would be item e that we consider, and item b would be eliminated, since not frequent in the remaining transactions.

- build tree containing e, then remove e
- remove all non frequent items

The FP-Tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).

Conditional FP-Tree

prefix tree used to count support of node. it tells us if we should continue building the conditional fp tree for e



conditional FP tree of ce would tell us that ace is infrequent.

Remember that the support is only updated AT the conditional tree, not the prefix tree

all conditional FP Trees that would be constructed from 2-itemsets ending in item e. Once the conditional FP Trees have been constructed, we can read off the support for all 2-itemsets, by traversing the lists at the leaf level (as we did first for item e itself). Once this step is finished, the algorithm continues in the same way for the remaining 2-itemsets. For example, for itemset de we would now construct a conditional FP Tree, by first extracting from tree (b) the tree with paths ending in de, resulting in tree (c), and then extracting the conditional FP Tree for itemset de (tree (d)).

- Check if leaves of the prefix path is frequent.
- Build prefix path of 2-itemsets or more from conditional FP tree but remember to update the counts

1. Build prefix path of node with smallest support / bottom most node
2. Count <leaf> and check if it is frequent
3. Build conditional FP tree of <leaf>
4. Build prefix path from of 3
5. Run 2
6. Repeat from 3

Result

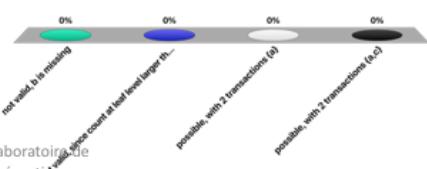
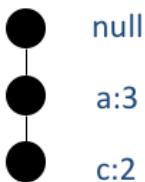
Frequent itemsets detected in this order

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

This table summarizes the frequent itemsets in the order they are detected, i.e., starting from the items with the lowest support.

The FP tree below is ...

- A. not valid, b is missing
- B. not valid, since count at leaf level larger than 1
- C. possible, with 2 transactions {a}
- D. possible, with 2 transactions {a,c}



©2018, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Summary FP Growth

Advantages

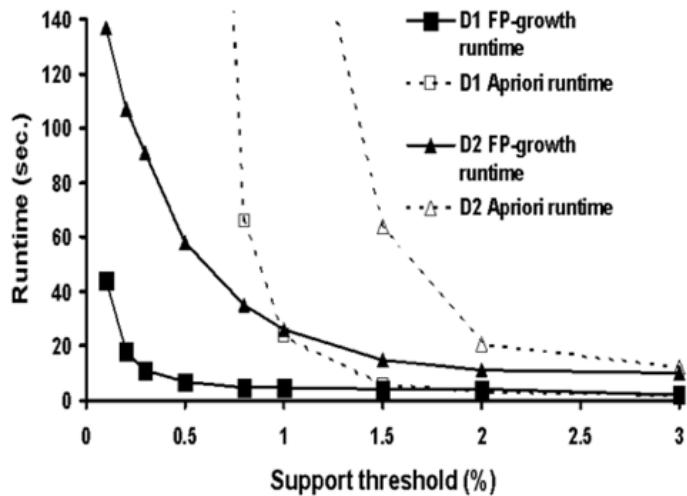
- Only 2 passes over the dataset
- Compresses dataset
- (Generally) much faster than Apriori

Disadvantages

- Works less efficiently for high support thresholds
- Has to run in main memory
- Difficult to find distributed implementation

FP Growth Works less efficiently for high support thresholds, because it prunes only single items.

Performance Comparison



Components of Data Analytics

1. Pattern structure/model representation
 - **association rules**
2. Scoring function
 - **support, confidence**
3. Optimisation and search
 - **JOIN, PRUNE**
 - **FP-Tree, ordering of items**
4. Data management
 - **transaction reduction, partitioning, sampling**

References

Textbook

- <http://www.mmds.org/mmds/v2.1/ch06-assocrules.pdf>
- Jiawei Han, *Data Mining: concepts and techniques*, Morgan Kaufman, 2000, ISBN 1-55860-489-8
- Pang-Ning Tan, Michael Steinbach, Vipin Kumar: *Introduction to Data Mining*, Addison-Wesley. Chapter 6: Association Analysis: Basic Concepts and Algorithms

Some relevant research literature

- R. Agrawal, T. Imielinski, and A. Swami. *Mining association rules between sets of items in large databases*. SIGMOD'93, 207-216, Washington, D.C.
- J. Han, J. Pei and Y. Yin. *Mining Frequent patterns without candidate generation*. SIGMOD 2000, 1–12.