User Modeling and Personalization 5: Web Personalization and Web Usage Mining

Eelco Herder

L3S Research Center / Leibniz University of Hanover
Hannover, Germany

2 May 2016

Outline I

- Web Personalization
- Web Usage Mining
  - Applications
  - Types of Data
  - Server-Side Data Collection
  - Data Preprocessing
  - Client-Side Data Collection
- Knowledge Inference
  - Predictive Models
  - A priori probabilities: ranking methods
  - (Conditional) probabilistic Models
- Assocation Rules
  - Support and Confidence

Outline II

Example: Supermarket

Association Rule Problem and Frequent Itemsets

Example Use of A Priori

Web Personalization

Web personalization is a specialization area of adaptive hypermedia, specifically aimed at improving the usability of Web sites.

Adaptive hypermedia systems build a model of the user and use this model throughout the interaction with the user.

The term ‘Web personalization’ is also often used in e-commerce, with a slightly different meaning and serving slightly different goals. In the field of e-commerce, Web personalization usually means the segmentation of user groups in order to create and aim effective messages at customers.

Stages of Web Personalization

Similar to the layers of the adaptation process, the process of Web personalization can be split into three stages:

- data collection and preprocessing;
- knowledge inference;
- personalization

Web Usage Mining

Web Usage Mining

Web usage mining is the process of discovering interesting usage patterns from the interactions of the users while surfing the Web.

Interesting patterns include:

- Frequently reoccurring actions
- Frequently visited pages
- Commonly followed paths
- Usage of search tools
- Strategies used to locate information

Although user tasks differ wildly, many regularities and useful patterns can be discovered.

Applications

Applications of Web usage mining include:

- understand user tasks and user needs;
- recognize usage patterns that indicate usability issues;
- target potential customers for ecommerce;
- enhance the quality and delivery of internet information services;
- improve Web server performance;
- ...

- identify potential advertisement locations;
- improve site design;
- fraud/intrusion detection;
- predict user's actions - for example for prefetching;
- personalization and adaptive sites.

Types of data

Four general categories of data sources for Web usage mining are distinguished:

Content data: the real data in Web pages, as presented to the end user.

- The data can be simple text, images or structured data, such as information retrieved from databases.
- Applications of content analysis include finding Web pages that contain similar content and determining which Web pages match a user query or user preferences;

Structure data: data that describes the organization of the content.
The data can be either data entities used within a Web page (*the content*), or data entities used to connect pages (*the navigation structure*) - most notably hyperlinks.

- Statistics on the content - such as the number of words, images or hyperlinks on a page - reflects the function of a Web page (e.g. does it mainly provide content or is it used for navigation).
- The navigation structure, as defined by connecting entities - such as hyperlinks - determines to a large extent user navigation patterns.

Usage data: data that describes the pattern of usage of Web pages, such as IP addresses, access date and time, referring pages and other attributes that can be retrieved

- From the raw Web usage data user groups may be distinguished, as well as groups of pages that are often visited together, popular pages and frequently followed paths.

User profile data: explicitly gathered data that provides demographic information about users of the Web site.

- This includes registration data and customer profile information.
- User profile data may help in explaining peculiarities in the Web usage data, but it can also be used as an additional source of information for personalization.

In this lecture, we focus on usage data.

Sources of Web Usage Data

Web Server

- Data from multiple users on one site
- In general limited to click-behavior

Web Client (i.e. Browser)

- Data from one user on multiple sites
- More sophisticated tracking (e.g. mouse movements)
- Requires user cooperation (e.g. installation of a program)

Proxy Server

- Data from multiple users on multiple sites
- Sits between server and client
- Analyses (and possibly modifies) the data stream

Server-Side Data Collection

The most widely used source of navigation data is the Web server. Most servers store all transactions in a *Web server log*. Whereas the Web Consortium has defined a standardized log format, several Web servers have their own proprietary format. Typical fields included in the log file are:

- The *requested URL*;
- The *remote IP* of the machine from which the page request originated. This may be a user's computer, a provider's proxy server or any other machine from which a page request can be made;
- A *timestamp* of the date and time of the page request;

- The *method* used for requesting the page;
- The *status code*, which indicates whether the page was retrieved successfully;
- The *number of bytes sent* to answer the request;
- The *referrer*: if the user followed a link, the originating url is listed here;
- The *user agent*, which indicates the browser used, or which robot requested the page.

Example server log entries:

(Already somewhat older)

- 213.6.31.68 - - [01/May/2004:22:38:32 +0200] "GET /forsale.html HTTP/1.1" 200 14956 "http://www.fortepiano.nl/indexforsale.html" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
- 213.6.31.68 - - [01/May/2004:22:38:34 +0200] "GET /pictures/forsale/bertsche1835-small.jpg HTTP/1.1" 200 5753 "http://www.fortepiano.nl/forsale.html" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"

Preprocessing web server data

Several issues need to be dealt with before the data from the Web server log can be analyzed:

Data cleaning. This step consists of removing all the requests that are not explicitly initiated by the user, such as additional files that are embedded in the originally requested page.

User identification. Whereas each computer has its own unique IP address, several providers use a proxy server to balance Web traffic. Heuristics or technical tricks are needed to separate users behind the proxy.
Further, sites such as search engines use *robots* for building and updating their database. The common politeness of having robots identifying themselves is not followed by all developers.

Session identification. It may be possible that users visit the site more than once. The goal of session identification is to divide the page accesses of each user into individual sessions.

Path completion. In practice, not all page requests reach the Web server. Browsers may store a page in their local cache for improving response time.
In addition, several providers use a proxy server that stores popular pages for improving response time as well as for reducing Web traffic.
A common approach in situations when two subsequent page visits to a and b are recorded, while no link between a and b exists, is to assume that a user backtracked from a to some page c, from which a link (b, c) exists.

Data cleaning

This step consists of removing all the requests that are not explicitly initiated by the user, such as additional files that are embedded in the originally requested page.

Typically, in this step graphical content such as jpg and gif images are removed.

A major artifact used to be formed by *framesets*, which break the page metaphor - what the user sees as a single page, may be constructed from multiple html files. As each html file causes an additional page request, multiple entries in the log file may relate to one user action. Due to the stateless character of the http protocol, it is impossible to identify which html files are loaded as part of one frameset.

Common heuristic: if an html file is loaded into an embedded window within a certain time interval (3 seconds) after a frameset is loaded, it isassumed to be part of the frameset.

User identification

One computer may be used by multiple users. One user may use multiple computers. ISPs may use a proxy server, so that multiple users have one IP address. IP addresses may be dynamic. So how to recognize a user?

- The easiest solution is to require users to **log in**. This might put users off, though.
- Another solution is to send a **cookie** to the user computer, or to use a unique session identifier.
- Should this not be possible, it might still be possible to distinguish users based on their **browser identifiers**.
- Should none of these methods be of any avail, one can analyze whether a requested page is reachable from the last requested page.

Robots

Most robots self-identify themselves in the server logs. If not, one can exploit the fact that robots explore the site more systematically and faster than users do, and return to the site at regular time intervals.

Robots explore the entire website in breadth first fashion

- Periodic Spikes (can overload a server)
- Lower-level constant stream of requests

Humans access web-pages in depth first fashion

- Daily pattern: Monday to Friday
- Hourly pattern: peak around midday, low traffic during nights

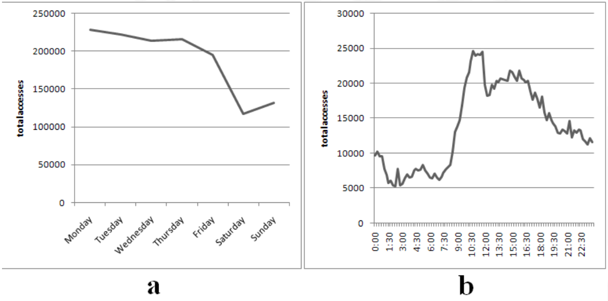


Figure: Typical human patterns of Web site usage

Session identification

Users do not access the Web continuously, but rather have several periods of increased navigation activity and periods in which they do not access the Web at all. These periods of increased navigation activity are assumed to represent a user task, or a set of user tasks.

Session identification - or rather session reconstruction - attempts to split the Web log into meaningful chunks, which are called sessions.

Whereas these sessions are related to the user activity, they are merely a technical concept, defined by the heuristics used during the reconstruction process.

The most common heuristic used for session reconstruction is a time-out mechanism; if the time between two subsequent page requests exceeds a certain time limit, it is assumed that the user is starting a new session.

Based on empirical data, the usual time-out is 25.5 minutes. This is based on an average time between two page requests, which was 9.3 minutes, and added 1.5 standard deviations to this number.

This is a standard statistical procedure to remove outliers from a dataset.

However, this approach only makes sense for normal distributions. Web browsing is a rapid activity in which users spend only little time on the far majority of pages. Nevertheless, based on the established 25.5 timeout, most researchers and commercial Web usage mining packages use this cutoff, in many cases rounded to 30 minutes.

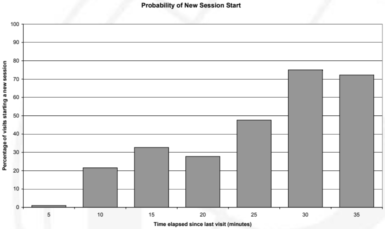


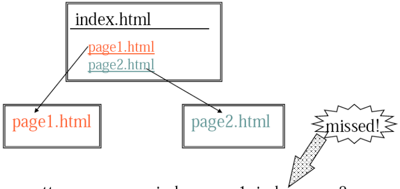
Figure: At 25 minutes there is a 50% chance that a next visit will initiate a new session.

Path completion

In practice, not all page requests reach the Web server. Browsers may store a page in their local cache for improving response time.

In addition, several providers use a proxy server that stores popular pages for improving response time as well as for reducing Web traffic.

A common approach in situations when two subsequent page visits to a and b are recorded, while no link between a and b exists, is to assume that a user backtracked from a to some page c, from which a link (b, c) exists.



Access pattern: index, page1, index, page2
Record in server log:: index, page1, page2

Proxies / Client Cache, from "Web Mining: Accomplishments & Future Directions" Jaideep Srivastava, University of Minnesota, pp 63-90. `http://www.ieee.org.ar/downloads/Srivastava-tut-pres.pdf`

Client-Side Data Collection

In contrast to Web server logs, *client side logging* provides data on user navigation on multiple servers. As the data is collected on the user computer, problems associated with user recognition and path completion are eliminated or reduced.

A popular early technique for client-side logging is the use of *instrumented browsers*. However, browser modification is not a trivial task for modern browsers.

The adoption of *browser plugins* (or browser extensions) is a recent alternative for instrumented browsers.

An alternative approach is the use of agents that are embedded in Web pages, for example as Java applets, which are used to collect information directly from the client.

A more intrusive method for client-side logging is the use of *spyware*.

Potentially, a large number of events can be recorded on client-side, from low-level events such as keystrokes and mouse clicks to higher-level events such as page requests.

There are several limitations to client-side logging:

- client-side modifications might alter the user's normal behavior
- users may be unable or unwilling to install special software that might cause problems or that might form a security thread
- some mechanism is needed to retrieve the data from the client computer

Knowledge Inference

The second stage is the inference of usable information from the observed data that can be used as an input for analysis or the Web personalization process.

Many mechanisms can be employed for this purpose, including:

- *graph theory*: mathematical analysis of the node-and-link structure of Web sites and navigation paths.
- *link analysis*: this method is closely related to graph-theoretic approaches, but link analysis includes more sophisticated methods for predicting the 'importance' of Web pages. The most well-known method is the PageRank system, originally used in the Google search engine;

- *probabilistic methods*: prediction of user navigation behavior, making use of stochastic models;
- *text analysis*: the indexing, scoring and categorization of textual documents, as well as discovering user interests making use of techniques developed in the fields of information retrieval and machine learning;
- *user segmentation*: the identification of groups of users, based on e.g. their demographics, background, interests, behavior and current user context.

Predictive Models

Predictive or probabilistic models of browsing behavior attempt to predict future navigation actions based on past observations. Probabilistic modeling can both generate insight in how we use the Web and provide mechanisms for making predictions that can be used for personalization.

The basic intuition behind predictive modeling is that some actions occur more often than other actions. The probability $P(a), 0 \leq P(a) \leq 1$ indicates the a priori likelihood of an action a. $P(a)$ will be high for actions that occur frequently and low for infrequent actions.

The likelihood of an action a will become higher if an action b is observed that tends to be followed by a. The conditional probability $P(a|b)$ represents the likelihood that a will happen given *evidence* that b occurred. In practice, this is the ratio between the number of times that b is followed by a and the number of times that b occurred.

Or, more generalized, the conditional probability $P(a|E)$ is the likelihood that action a will occur, given some evidence E. Bayes' theorem (1) is used for updating the conditional probability:

$$P(a|E) = \frac{P(E|a)P(a)}{P(E)} \tag{1}$$

A priori probabilities: ranking methods

The aim of ranking methods is to provide for each item a numerical estimation of the *a priori* likelihood that it will be accessed in the next transaction.

After each page request, the selected ranking method goes through all visited items of interest (either pages or sites), estimates their value and sorts them in descending order of their expected value.

Ranking methods make use of the typical distribution of revisited pages:

- A small number of pages is visited very frequently, a large number of pages is visited very infrequently
- Most pages that are revisited have been visited quite recently the longer the time has passed, the less likely it is that a page will be revisited.

This distribution is called a *power law* and can be observed in many cases.

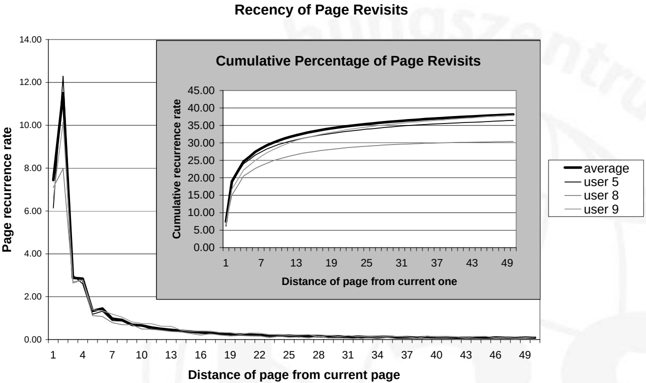


Figure: Percentage of page revisits as a function of distance from the current page. The popularity distribution shows a similar pattern.

These distributions can be directly applied for the following (naive) ranking methods for all pages m_i in the navigation history:

- Last Recently Used (**LRU**),

$$LRU(m_i, I_{m_i}, i_n) = \frac{1}{i_n - i_k + 1},$$

- Most Frequently Used (**MFU**),

$$MFU(m_i, I_{m_i}, i_n) = \frac{|I_{m_i}|}{i_n},$$

where

- $I = i_1 \ldots i_n$ is the (numbered) list of page visits in the navigation history
- m_i is the page identifier and I_{m_i} (subset of I) the visits to this page
- $|I_{m_i}|$ the cardinality of (i.e. the number of items within) I_{m_i}
- i_n the index of the last page request in the navigation history I
- i_k the last visit to page m_i (i.e. the index of the last page request in I_{m_i})

An example method that combines recency and frequency is Polynomial decay:

- Polynomial Decay (**PD**),

$$DEC(m_i, I_{m_i}, i_n) = \sum_{j=1}^{|I_{m_i}|} \frac{1}{1 + (i_n - i_j)^{\alpha}},\ 0 < \alpha \leq 1$$

In words, PD assigns to each page a rank that is the sum of previous visits to this page; visits from the more distant past get lower weighting.
α controls to what extent visits from the more distant past are taken into account:

- a high value for α emphasizes recency
- a low value for α emphasizes frequency

(Conditional) probabilistic Models

Basic idea: The likelihood of an action a will become higher if an action b is observed that tends to be followed by a.

Markov Models

A straightforward probabilistic model for page prediction is the (first order) Markov model. Under a first-order Markov assumption, it is assumed that the probability of each state s_t given the preceding states $s_{t-1}, \ldots, s_1$ only depends on the last state s_{t-1}:

$$P(s_n) = P(s_1) \prod_{t=2}^{n} P(s_t|s_{t-1}) \tag{2}$$

It can be shown that the probability of a state transition t_{ij} from s_i to s_j equals to the ratio between the number of transitions from s_i to s_j, n_{ij} and the total number of transitions from s_i, n_i observed thus far.

$$P(t_{ij}) = \frac{n_{ij}}{n_i} \tag{3}$$

A (first-order) Matrix model can be represented by a simple *transition matrix*:

From / to	A	B	C	D	Total
A		3	5	8	16
B	3		7	4	14
C	2	4		6	12
D	1	6	2		9

Various variations of first order Markov models have been proposed in the literature. The most common generalization is to use k-th order Markov models, which assume that the probability of each next state depends on the last k states.

In practice it is infeasible to use a simple Markov model for predicting page visits within Web sites, as the number of pages in a typical Web site is quite large. For this reason, in most applications the pages are grouped in a number of categories; in this case, the Markov model would predict which page category will be visited next.

In more detail: Association Rules

Association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases.

Association Rule

Given a set of items $I = \{I_1, I_2, \ldots, I_m\}$ and a database with transactions $D = \{t_1, t_2, \ldots, t_n\}$, with $t_i = \{I_{i1}, I_{i2}, \ldots I_{ik}\}$ and $I_{ij} \in I$. An association rule is an implication $X \Longrightarrow Y$; $X, Y \subset I$ are item sets and $X \cap Y = \emptyset$.

We are only interested in association rules that are sufficiently backed up by the data. In other words: the item sets X and Y should occur *often enough* together in a transaction and there should not be too many transactions in which X occurs, but not Y.

Measures used to verify these properties are **Support** and **Confidence**.

Support

The **Support (s)** of an Association Rule $X \Longrightarrow Y$ is the percentage of transactions in database D that contain the items $X \cup Y$:

$$\textbf{support}(X \Longrightarrow Y) = \frac{|\{t_i \in D : X \cup Y \subset t_i\}|}{|D|}$$

Confidence

The **Confidence (α)** in an Association Rule $X \implies Y$ is the ratio between the transactions that contain $X \cup Y$ and the transactions that only contain X:

$$\textbf{confidence}(X \implies Y) = \frac{|\{t_i \in D : X \cup Y \subset t_i\}|}{|\{t_i \in D : X \subset t_i\}|}$$

Example: Supermarkt

An (online) supermarket wants to analyze the weekly purchases of its customers. In order to do this, the following steps are followed:

1. Data acquisition and preparation
   - Acquiring the purchase logs
   - Removing returned items from the logs
   - Identifying individual customers and transactions (purchases)
2. Knowledge inference
   - Goal: which products are often bought together?
   - Method used: association rules
3. Optimization and personalization
   - identify thematic groups of products and place them together
   - recommend products and deals that fit the customers' profiles
   - create bundled backages

Example data for the supermarket

Transaction	Bought Items
t1	Bread, Jelly, Peanut butter
t2	Bread, Peanut butter
t3	Bread, Milk, Peanut butter
t4	Beer, Bread
t5	Beer, Milk

Set	Support	Set	Support
Beer	40	Beer, Bread, Milk	0
Bread	80	Beer, Bread, Peanut-butter	0
Jelly	20	Beer, Jelly, Milk	0
Milk	40	Beer, Jelly, Peanut-butter	0
Peanut-butter	60	Beer, Milk, Peanut-butter	0
Beer, Bread	20	Bread, Jelly, Milk	0
Beer, Jelly	0	Bread, Jelly, Peanut-butter	20
Beer, Milk	20	Bread, Milk, Peanut-butter	20
Beer, Peanut-butter	0	Jelly, Milk, Peanut-butter	0
Bread, Jelly	20	Beer, Bread, Jelly, Peanut-butter	0
Bread, Milk	20	Beer, Bread, Milk, Peanut-butter	0
Bread, Peanut-butter	60	Beer, Bread, Jelly, Milk	0
Jelly, Milk	0	Beer, Jelly, Milk, Peanut-butter	0
Jelly, Peanut-butter	20	Bread, Jelly, Milk, Peanut-butter	0
Milk, Peanut-butter	20	Beer, Bread, Jelly, Milk, Peanut-butter	0
Beer, Bread, Jelly	0		

Support and Confidence

For Association Rules from the Supermarket Example

$X \Longrightarrow Y$	s	α
Bread $\Longrightarrow$ Peanut-Butter	60%	75%
Peanut-Butter $\Longrightarrow$ Bread	60%	100%
Beer $\Longrightarrow$ Bread	20%	50%
Peanut-Butter $\Longrightarrow$ Jelly	20%	33.3%
Jelly $\Longrightarrow$ Peanut-Butter	20%	100%
Jelly $\Longrightarrow$ Milk	0%	0%

Discussion

Support indicates how often a rule occurs in the database.

As the database probably contains many transactions involving other products than the ones in the rule, it is not uncommon that support values are quite low

- But still, there needs to be a lower limit that excludes rules based on only one or a small number of transactions).

By contrast, confidence indicates how good a rule is. Therefore, values for confidence should be rather high.

Example:

- Jelly $\Longrightarrow$ Peanut Butter: Support: 20%, Confidence: 100%
- This rule covers 20% of the transactions
- But whenever someone buys jelly, he will buy peanut butter as well

Association Rule Problem

Given a set of items $I = \{I_1, I_2, \ldots, I_m\}$ and a database with transactions $D = \{t_1, t_2, \ldots, t_n\}$ with $t_i = \{I_{i1}, I_{i2}, \ldots I_{ik}\}$ and $I_{ij} \in I$.
The Association Rule Problem involves finding all Association Rules $X \implies Y$ with a support larger than a given minimum support level s and a confidence larger than a given minimum confidence level α.

The most difficult part of this problem is to find all **frequent itemsets** X and Y.

Frequent Itemsets

An *itemset* is a subset of all items I. A *frequent itemset* is an itemset that occurs more often than a given minimum level of support s.

The finding of frequent itemsets is important, because we know that for each association rule $X \Longrightarrow Y$ yields that $X \cup Y$ is part of these frequent itemsets.

genRules

Algorithm to determine association rules if frequent item sets are given:

```
Input:
   D       // database of transactions
   I       // items
   F       // Set of frequent itemsets
   s       // min_support
   alpha   // min_confidence

Output:
   R       // Association rules satisfying s and alpha
```

```
Algorithm:
   R = emptyset:
   for each element f of F do
       for each non-empty subset x of f
          if support(f) / support(x) >= alpha then
          // does the rule satisfy the minimum confidence
             add x => f\x  to R.
```

How to find association rules if frequent itemsets are given?

With min_support = 30%, and alpha = 50%.
Frequent itemsets:

{ {*Beer*}, {*Bread*}, {*Milk*}, {*Peanut-butter*},
{*Bread,Peanut-Butter*} }

You can check in the table with transaction data that all itemsets have a support larger than 30%.

Result: R={Bread $\Longrightarrow$ Peanut-butter, Peanut-Butter $\Longrightarrow$ Bread}

- the first rule has a confidence of $\frac{3}{4} = 75\%$
- the second rule has a confidence of $\frac{3}{3} = 100\%$

Finding frequent itemsets

It is not hard to find frequent itemsets.

A naive approach would be to check the Support of all possible combinations of items. That would mean that for a list with m items, one would need to check $2^m - 1$ possible subsets.

A more structured approach is given by the a priori algorithm.

Apriori Algorithm

The apriori algorithm is the most well-known algorithm to determine frequent itemsets

It exploits the following property:

- **Downward Closure**: each subset of a frequent itemset is a frequent itemset itself

Or, the other way round: if an itemset is not a frequent itemset, then it cannot be a subset of a frequent itemset.

Main idea of the Apriori Algorithm

- Generate a set of *candidate itemsets* of a certain size $s-1$
- Remove all candidate itemsets that do not have sufficient support
- Iteratively merge all remaining frequent itemsets that have all except one items in common to a larger candidate itemset of size s

Generate Candidates

The Apriori-Algorithm uses Apriori-GenerateCandidates.The following algorithm generates candidates for frequent itemsets of size i.

It requires as an input all frequent itemsets of size (i-1) and merges all sets that differ in exactly one element.

Generate Candidates Algorithm

```
Input: F(i-1)   // frequent itemsets of size i-1
Output: C(i)   // candidates for frequent itemsets
                    // of size i

GenerateCandidates
C(i) = emptyset
for each element I of F(i-1) do  // for each frequent
                          //  itemset of size i-1
  for each element J of F(i-1), J unequal I,
    if i-2 of the elements in I and J are similar then
                          // if I and J differ in exactly
                           // one element
       C(i) = C(i) U {I U J}
```

Apriori Algorithm

- The algorithm starts with the set $C(1)$ of all single items.
- All items with a minimum support are added to the frequent itemset $F(k)$ of length k.
- $F(k)$ is used by GenerateCandidates for generating candidate set $C(k+1)$.
- The Apriori algorithm is iteratively repeated for $k+1$, until no frequent itemsets are found.

Apriori

```
Input: {i1, i2, .. in}  // set of all items
       D      // Database of transactions
       s      // minimum support
Output: F     // frequent itemsets

Apriori-Algorithm:
k = 0;        // k is used as the scan number
F = emptyset
C(1) = {i1, i2, .. in};
        // C(i): candidate itemsets of size i
       // initial candidates are set to be the items
```

```
repeat
  k = k+1;
  F(k) = emptyset;   // F(K): frequent itemsets of size k

  for each element I(i) of C(k) do
                     // for each itemset in C(k)
     c(i)=0;    // initial count is set to 0.

  for each transaction t in D do
     for each element I(i) of C(k) do
        if I(i) in t then
           c(i) = c(i) + 1;     // if an itemset occurs in
           // a transaction, we increment its count by 1
```

```
 for each I(i) in C(k) do
 if s <= c(i) / |D| then    // if the itemset is frequent
        F(k) = F(k) U I(i)      // we add it to the
                   //  frequent itemsets of size k

 C(k+1) = Apriori-GenerateCandidates( F(k) )
           // generate candidates for frequent itemsets
           // of size k+1

F = F U F(k)     // add F(k) to the frequent itemsets
until C(k+1) = emptyset   // until we do not find
                    // any itemsets of size (k+1)
```

Example Use of A Priori

Example Data for Grocery Store:

Transaction	Items
t1	Bread, Jelly, Peanut-Butter
t2	Bread, Peanut-Butter
t3	Bread, Milk, Peanut-Butter
t4	Beer, Bread
t5	Beer, Milk

Using Apriori algorithm to determine frequent itemsets (s=30%):

Pass	Candidates	Frequent Itemsets

1	{Beer},{Bread},{Jelly},
	{Milk}, {Peanutbutter}
		{Beer}, {Bread}
		{Milk}, {Peanutbutter}

2	{Beer,Bread}, {Beer,Milk}
	{Beer,Peanutbutter}, {Bread,Milk}
	{Bread,Peanutbutter}, {Milk,Peanutbutter}
		{Bread,Peanutbutter}

Clothing Store

Example Data for Clothing Store

Transaction	Items	Transactions	Items
t1	Blouse	t11	T-Shirt
t2	Coat,Pullover,T-Shirt	t12	Blouse,Jeans,Coat,Pullover,T-Shirt
t3	Jeans,T-Shirt	t13	Jeans,Coat,Shorts,T-Shirt
t4	Jeans,Coat,T-Shirt	t14	Coat,Pullover,T-Shirt
t5	Jeans,Shorts	t15	Jeans,T-Shirt
t6	Coat,T-Shirt	t16	Pullover,T-Shirt
t7	Jeans,Pullover	t17	Blouse,Jeans,Pullover
t8	Jeans,Coat,Shorts,T-Shirt	t18	Jeans,Coat,Shorts,T-Shirt
t9	Jeans	t19	Jeans
t10	Jeans,Coat,T-Shirt	t20	Jeans,Coat,Shorts,T-Shirt

Clothing Store

Using Apriori algorithm to determine frequent itemsets (s=20%):

Pass	Candidates	Frequent Itemsets
1	{Blouse},{Jeans},{Coat}, {Shorts},{Pullover},{T-Shirt}	{Jeans}, {Coat},{Shorts}, {Pullover}, {T-Shirt}
2	{Jeans,Coat},{Jeans,Shorts},{Jeans,Pullover}, {Jeans,T-Shirt},{Coat,Shorts},{Coat,Pullover}, {Coat,T-Shirt},{Shorts,Pullover},{Shorts,T-Shirt}, {Pullover,T-Shirt}	{Jeans,Coat}, {Jeans,Shorts}, {Jeans,T-Shirt}, {Coat,Shorts}, {Coat,T-Shirt}, {Shorts,T-Shirt}, {Pullover,T-Shirt}

3	{Jeans,Coat,Shorts},{Jeans,Coat,T-Shirt}
	{Jeans,Shorts,T-Shirt},{Jeans,Pullover,T-Shirt}
	{Coat,Shorts,T-Shirt},{Coat,Pullover,T-Shirt}
	{Shorts,Pullover,T-Shirt}
		{Jeans,Coat,Shorts},
		{Jeans,Coat,T-Shirt},
		{Jeans,Shorts,T-Shirt},
		{Coat,Shorts,T-Shirt}

4	{Jeans,Coat,Shorts,T-Shirt}	-