Create account Log in

Article Talk Read Edit View history Search

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
فارسی
Français
Italiano
Nederlands
Português
Русский
Српски / srpski
中文
Edit links

Apriori algorithm

From Wikipedia, the free encyclopedia

This article **may be confusing or unclear to readers**. Please help us clarify the article; suggestions may be found on the talk page. *(December 2006)*

Apriori[1] is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

Contents [hide]

Overview [edit]

The Apriori algorithm was proposed by Agarwal and Srikant in 1994. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing). Each transaction is seen as a set of items (an *itemset*). Given a threshold C, the Apriori algorithm identifies the item sets which are subsets of at least C transactions in the database.

Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation*), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k-1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

The pseudo code for the algorithm is given below for a transaction database T, and a support threshold of ϵ. Usual set theoretic notation is employed, though note that T is a multiset. C_k is the candidate set for level k. At each step, the algorithm is assumed to generate the candidate sets from the large item sets of the preceding level, heeding the downward closure lemma. $count[c]$ accesses a field of the data structure that represents candidate set c, which is initially assumed to be zero. Many details are omitted below, usually the most important part of the implementation is the data structure used for storing the candidate sets, and counting their frequencies.

```
$\mathrm{Apriori}(T, \epsilon)$
    $L_1 \leftarrow \{\mathrm{large}\ 1 - \mathrm{itemsets}\}$
    $k \leftarrow 2$
    while $L_{k-1} \neq \emptyset$
        $C_k \leftarrow \{a \cup \{b\} \mid a \in L_{k-1} \land b \not\in a\} - \{c \mid \{s \mid s \subseteq c \land |s| = k-1\} \nsubseteq L_{k-1}\}$
        for transactions $t \in T$
            $C_t \leftarrow \{c \mid c \in C_k \land c \subseteq t\}$
            for candidates $c \in C_t$
                $count[c] \leftarrow count[c] + 1$
        $L_k \leftarrow \{c \mid c \in C_k \land\ count[c] \geq \epsilon\}$
        $k \leftarrow k + 1$
    return $\bigcup_k L_k$
```

Examples [edit]

Example 1 [edit]

Consider the following database, where each row is a transaction and each cell is an individual item of the transaction:

alpha	beta	epsilon
alpha	beta	theta
alpha	beta	epsilon
alpha	beta	theta

The association rules that can be determined from this database are the following:

1. 100% of sets with alpha also contain beta
2. 50% of sets with alpha, beta also have epsilon
3. 50% of sets with alpha, beta also have theta

we can also illustrate this through a variety of examples

Example 2 [edit]

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item: each item, such as "butter" or "bread", is identified by a numerical SKU. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together.

Let the database of transactions consist of following itemsets:

Itemsets
{1,2,3,4}
{1,2,4}
{1,2}
{2,3,4}
{2,3}
{3,4}
{2,4}

We will use Apriori to determine the frequent item sets of this database. To do so, we will say that an item set is frequent if it appears in at least 3 transactions of the database: the value 3 is the *support threshold*.

The first step of Apriori is to count up the number of occurrences, called the support, of each member item separately, by scanning the database a first time. We obtain the following result

Item	Support

{1}	3
{2}	6
{3}	4
{4}	5

All the itemsets of size 1 have a support of at least 3, so they are all frequent.

The next step is to generate a list of all pairs of the frequent items:

Item	Support
{1,2}	3
{1,3}	1
{1,4}	2
{2,3}	3
{2,4}	4
{3,4}	3

The pairs {1,2}, {2,3}, {2,4}, and {3,4} all meet or exceed the minimum support of 3, so they are frequent. The pairs {1,3} and {1,4} are not. Now, because {1,3} and {1,4} are not frequent, any larger set which contains {1,3} or {1,4} cannot be frequent. In this way, we can *prune* sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

Item	Support
{2,3,4}	2

In the example, there are no frequent triplets -- {2,3,4} is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold.

We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because one of their subsets was already known to be below the threshold.

Limitations [edit]

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|} - 1$ of its proper subsets.

Later algorithms such as Max-Miner[2] try to identify the maximal frequent item sets without enumerating their subsets, and perform "jumps" in the search space rather than a purely bottom-up approach.

References [edit]

1. ^ Rakesh Agrawal and Ramakrishnan Srikant Fast algorithms for mining association rules in large databases. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.
2. ^ Bayardo Jr, Roberto J. (1998). "Efficiently mining long patterns from databases" (PDF). *ACM Sigmod Record.* **27** (2).

External links [edit]

- ARtool, GPL Java association rule mining application with GUI, offering implementations of multiple algorithms for discovery of frequent patterns and extraction of association rules (includes Apriori)
- ELKI includes Java implementations of Apriori, Eclat and FPGrowth.
- SPMF offers Java open-source implementations of Apriori and several variations such as AprioriClose, UApriori, AprioriInverse, AprioriRare, MSApriori, AprioriTID, and other more efficient algorithms such as FPGrowth and LCM.
- Christian Borgelt provides C implementations for Apriori and many other frequent pattern mining algorithms (Eclat, FPGrowth, etc.). The code is distributed as free software under the MIT license.
- The R package arules contains Apriori and Eclat and infrastructure for representing, manipulating and analyzing transaction data and patterns.

Categories: Data mining algorithms