tu dortmund university dm

🏠 › Machine Learning Bits › Apriori Algorithm

Apriori Algorithm

A previous ▶ YouTube video recording of these contents is available.

Apriori Algorithm [AgSr94]

Algorithm: Apriori(I, D, $minsupp$)

$k \leftarrow 1$
$F_1 \leftarrow \{\{i\} \mid i \in I \wedge count(\{i\}) \geq |D| \cdot minsupp\}$ // frequent 1-itemsets
while $F_k \neq \emptyset$ **do**
  $k \leftarrow k+1$
  $C_k \leftarrow$ apriori-gen(F_{k-1}) // generate new candidates
  foreach $t \in D$ **do** // scan the database once
    $C_t \leftarrow$ subset(C_k, t) // candidates contained in t
    foreach $c \in C_t$ **do** c.count += 1 // increment support count
  $F_k \leftarrow \{c \in C_k \mid c.\text{count} \geq |D| \cdot minsupp\}$ // keep frequent candidates
return $\bigcup_i F_i$

where C_k: set of candidate k-itemsets F_k: set of frequent k-itemsets

Note: all itemsets & lists of itemsets are kept sorted.

Candidate Generation: apriori-gen

Step 1: Join by prefix matching

Process all $(k-1)$-frequent itemsets $p < q$, that *match* on their first $k-2$ elements

$$p \in F_{k-1}: (1\ 2\ 3)$$
$$\downarrow$$
$$(1\ 2\ 3\ 4) \in C_k$$
$$\uparrow$$
$$q \in F_{k-1}: (1\ 2\ 4)$$

Step 2: **Pruning**

Remove all candidate itemsets that have a subset of size (k - 1) that is not in F_{k-1}

Note: faster on *sorted* itemsets. Need to check only the k – 2 itemsets *not* used in join.

Example:	F_3 = {(1 2 3), (1 2 4), (1 3 4), (1 3 5), (2 3 4)}
After join:	{(1 2 3 4), (1 3 4 5)}
pruning step:	remove (1 3 4 5) because (3 4 5), (1 4 5) $\notin F_3$
⇝	C_4 = {(1 2 3 4)}

Apriori: Example with *minsupp* = 50%

TID	Items
1	A B C D E
2	A B E
3	A B D E
4	A B D
5	B C D E
6	B D E
7	A B C E
8	C D
9	A B D
10	A B D

scan D ⟶ C_1: { A } 7, { B } 9, { C } 4, { D } 8, { E } 6

filter ⟶ F_1: { A } 7, { B } 9, { D } 8, { E } 6

join & prune ⟶ C_2: { A B }, { A D }, { A E }, { B D }, { B E }, { D E }

scan D ⟶ C_2: { A B } 7, { A D } 5, { A E } 4, { B D } 7, { B E } 6, { D E } 4

filter ⟶ F_2: { A B } 7, { A D } 5, { B D } 7, { B E } 6

join ⟶ C_3: { A B D }, { B D E }

prune ⟶ C_3: { A B D }

scan D ⟶ C_3: { A B D } 5

filter ⟶ F_3: { A B D } 5

join ⟶ C_4: empty

References

[AgSr94] Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules in large databases. *VLDB* (1994), 487–499.

› Machine Learning Bits › The Apriori Hash Tree

The Apriori Hash Tree

A previous ▶ YouTube video recording of these contents is available.

Efficient Computing of the Subset Function

Subset(C_k,T) computes all candidates from C_k that are contained in transaction T

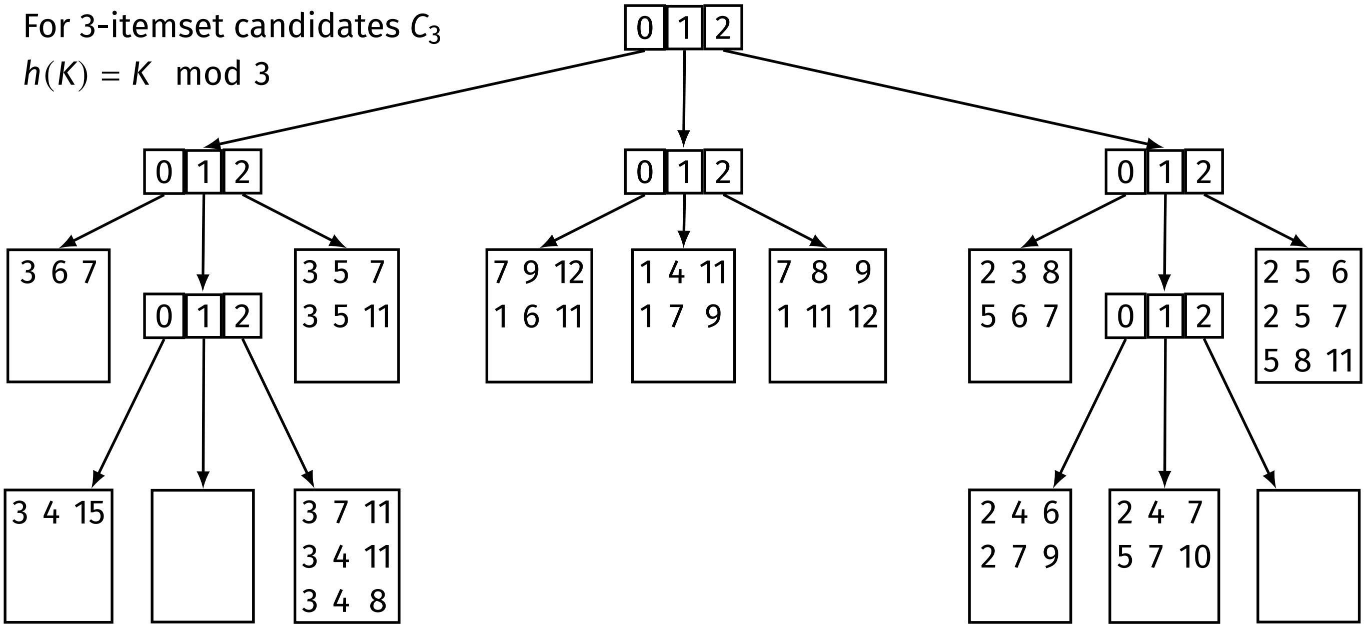
Problems:

- many candidate itemsets (millions)
- a transaction may contain many items / itemsets

Idea: use a hashtree to store/index/manage C_k (and use it to process each transaction)

- leaf nodes contain list of itemsets (including their counts, initially 0)
- inner nodes hold hash tables;
  each bucket at level d refers to a child node at level d + 1
- root node is at level 1
- hashing at level d is based on the dth smallest value in the k-itemset; at most k + 1 levels

Example



Hash Tree Index

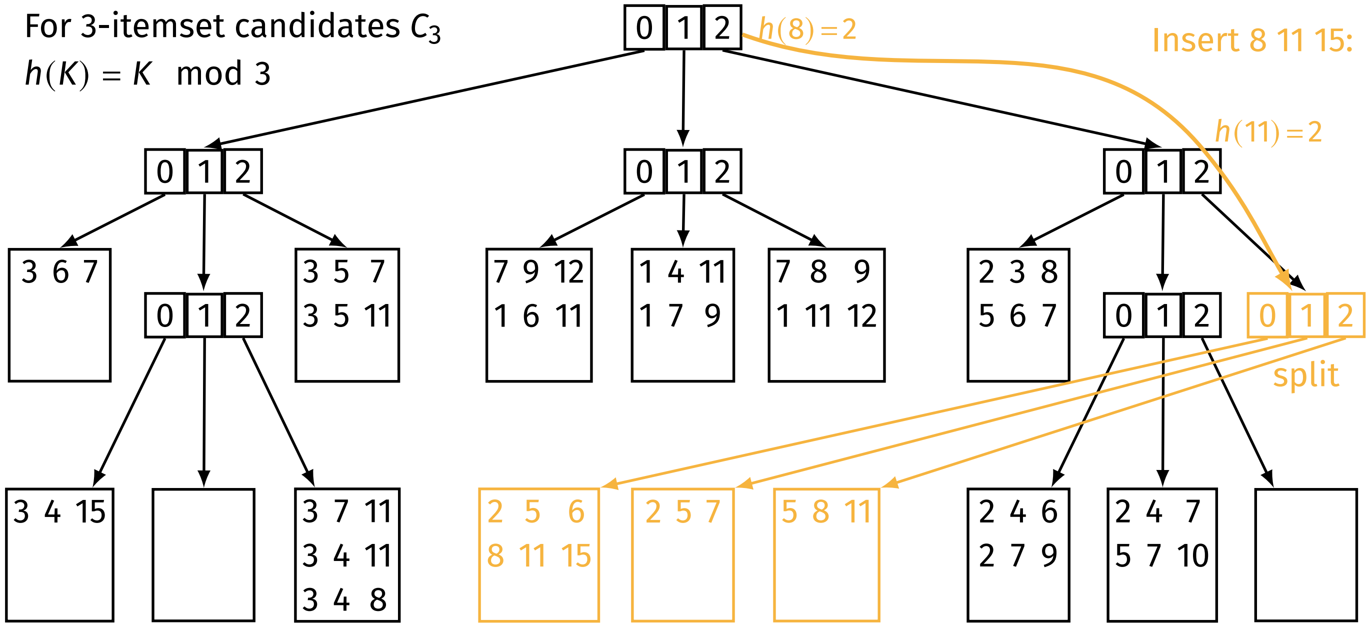
Search for a single *k*-itemset

- begin at the root node
- in a leaf: search linearly (or, e.g., binary search)
- inner node at level *d*: apply the hash function *h* to *d*-th item of the itemset to determine the branch to follow

Insertion of a *k*-itemset

- search for corresponding leaf node and insert itemset into node
- in case of a node overflow:
  - transform leaf node into inner node
  - distribute the node's entries to new leaf nodes according to hash function

Example: Search and Insert



Determining Frequent Itemsets using Hash Tree

Search all candidates that are contained in transaction $T = (t_1 t_2 \ldots t_m)$

at the root node:

- determine the hash value for *each* item in T
- continue search in each corresponding child node

at an inner node at level d (which has been reached by hashing t_i)

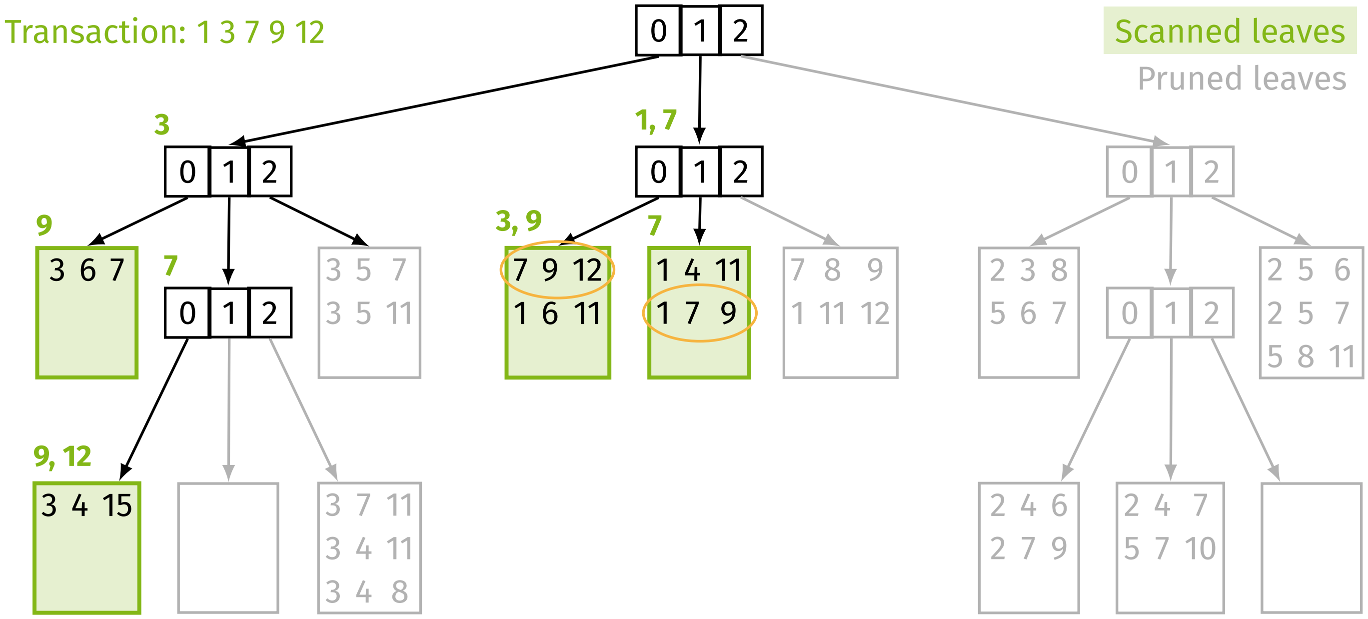
- determine hash values and continue search for each item t_k with $k > i$

at a leaf node

- test whether the itemsets in that node are contained in transaction T

Optimization: at every level d, only consider the hash codes of $(t_{i+1}, \ldots, t_{m-k+d})$ as possible next-smallest item, where t_i was the item last used for splitting.

Example



Only 6 of 22 candidates from C_3 are considered for this transaction, 2 of which are correct.

« NetFlix Example

Association Rules »

Technische Universität Dortmund
Informatik VIII
AG Data Mining

Otto-Hahn-Straße 14
44227 Dortmund

Telefon: (+49) 231 755-6391
Send Email