The Apriori Algorithm for Finding Association Rules

function apriori \((I, T, s_{\text{min}}, c_{\text{min}}, k_{\text{max}})\) begin
\[
k := 1;
\]
\[
C_k := \bigcup_{i\in I} \{i\};
\]
\[
F_k := \text{prune}(C_k, T, s_{\text{min}});
\]
while \(F_k \neq \emptyset\) and \(k \leq k_{\text{max}}\) do begin
\[
C_{k+1} := \text{candiates}(F_k);
\]
\[
F_{k+1} := \text{prune}(C_{k+1}, T, s_{\text{min}});
\]
\[
k := k + 1;
\]
end;
\[
R := \emptyset;
\]
forall \(f \in \bigcup_{j=2}^{k} F_j\) do begin
\[
m := 1;
\]
\[
H_m := \bigcup_{i\in f} \{i\};
\]
repeat
\[
\text{forall } h \in H_m \text{ do }
\]
if \(\frac{n(h)}{m} \geq c_{\text{min}}\) then \(R := R \cup \{(f-h) \rightarrow h\}\); else \(H_m := H_m - \{h\}\);
\[
H_{m+1} := \text{candidates}(H_m);
\]
\[
m := m + 1;
\]
until \(H_m = \emptyset\) or \(m \geq |f|\);
end;
\[
\text{return } R;
\]
end (* apriori *)

function candidates \((F_k)\) begin
\[
C := \emptyset;
\]
forall \(f_1, f_2 \in F_k\) with \(f_1 = \{i_1, \ldots, i_{k-1}, i_k\}\) and \(f_2 = \{i_1, \ldots, i_{k-1}, i'_k\}\) and \(i_k < i'_k\) do begin
\[
f := f_1 \cup f_2 = \{i_1, \ldots, i_{k-1}, i_k, i'_k\};
\]
if \(\forall i \in f : f - \{i\} \in F_k\) then \(C := C \cup \{f\}\);
end;
\[
\text{return } C;
\]
end (* candidates *)

function prune \((C, T, s_{\text{min}})\) begin
forall \(c \in C\) do
\[
s(c) := 0;
\]
forall \(t \in T\) do
forall \(c \in C\) do
if \(c \in t\) then \(s(c) := s(c) + 1\);
\[
F := \emptyset;
\]
forall \(e \in E\) do
if \(s_T(c) \geq s_{\text{min}}\) then \(F := F \cup \{e\}\);
\[
\text{return } F;
\]
end (* prune *)

(* apriori algorithm for association rules *)
(* — find frequent item sets *)
(* start with single element sets *)
(* and determine the frequent ones *)
(* while there are frequent item sets *)
(* create item sets with one item more *)
(* and determine the frequent ones *)
(* increment the item counter *)

(* — generate association rules *)
(* traverse the frequent item sets *)
(* start with rule heads (consequents) *)
(* that contain only one item *)
(* traverse rule heads of increasing size *)
(* traverse the possible rule heads *)
(* if the confidence of the rule *)
(* is high enough, add it to the result, *)
(* otherwise discard the rule head *)
(* create rule heads with one item more *)
(* increment the head item counter *)
(* until there are no more rule heads *)
(* or the antecedent would become empty *)
(* return the rules found *)

(* generate candidates with \(k + 1\) items *)
(* initialize the set of candidates *)
(* traverse all pairs of frequent item sets *)
(* that differ only in one item and *)
(* are in a lexicographic order *)
(* the order is arbitrary, but fixed *)
(* the union of these sets has \(k + 1\) items *)
(* only if all \(k\) element subsets are frequent, *)
(* add the new item set to the candidates *)
(* otherwise it cannot be frequent *)
(* return the generated candidates *)

(* prune infrequent candidates *)
(* initialize the support counters *)
(* of all candidates to be checked *)
(* traverse the transactions *)
(* traverse the candidates *)
(* if the transaction contains the candidate, *)
(* increment the support counter *)
(* initialize the set of frequent candidates *)
(* traverse the candidates *)
(* if a candidate is frequent, *)
(* add it to the set of frequent item sets *)
(* return the pruned set of candidates *)