Pincer-Search Algorithm for Discovering Maximum Frequent Set

Notes by Akash Saxena
(NITJ)

Introduction

Pincer Search Algorithm was proposed by, Dao-I Lin & Zvi M. Kedem of New York University in 1997. This algorithm uses both, the top-down and bottom-up approaches to Association Rule mining. It is a slight modification to Original Apriori Algorithm by R. Aggarwal & Srikant. In this the main search direction is bottom-up (same as Apriori) except that it conducts simultaneously a restricted top-down search, which basically is used to maintain another data structure called Maximum Frequent Candidate Set (MFCS). As output it produces the Maximum Frequent Set i.e. the set containing all maximal frequent itemsets, which therefore specifies immediately all frequent itemsets. The algorithm specializes in dealing with maximal frequent itemsets of large length. The authors got their inspiration from the notion of *version space* in Mitchell's machine learning paper.

Concepts used:

Let $I = (i_1, i_2, \ldots, i_m)$ be a set of *m* distinct items.

Transaction: A transaction T is defined as any subset of items in I.

Database: A database D is a set of transactions.

Itemset: A set of items is called an Itemset.

Length of Itemset: is the number of items in the itemset. Itemsets of length 'k' are referred to as k-itemsets.

Support Count: It is the total frequency of appearance of a particular pattern in a database.
In other terms: If T is a transaction in database D and X is an itemset then T is said to *support* X, if $X \subseteq T$, hence support of X is the fraction of transactions that support X.

Frequent Itemset: An itemset whose support count is greater than or equal to the minimum support threshold specified by the user.

Infrequent Itemset: An itemset which is not frequent is infrequent.

Downward Closure Property (Basis for Top-down Search): states that "If an itemset is frequent then all its subsets must be frequent."

Upward Closure Property (Basis for Bottom-up Search): states that "If an itemset is infrequent then all its supersets must be infrequent."

Maximal Frequent Set: it is a set, which is frequent and so are all its proper subsets. All its proper supersets are infrequent.

Maximum Frequent Set: is the set of all Maximal Frequent sets.

Association Rule: is the rule of the form R : $X \rightarrow Y$, where X and Y are two non-empty and non-intersecting itemsets. *Support* for rule R is defined as support of $X \cup Y$. ***Confidence*** is defined as Support of $X \cup Y$ / Support of X.

Interesting Association Rule: An association rule is said to be interesting if its support and confidence measures are equal to or greater than minimum support and confidence thresholds (specified by user) respectively.

Candidate Itemsets: It is a set of itemsets, which are to be tested to determine whether they are frequent or infrequent.

Maximum Frequent Candidate Set (MFCS): is a minimum cardinality set of itemsets such that the union of all the subsets of elements contains all frequent itemsets but does not contain any infrequent itemset, i.e. it is a minimum cardinality set satisfying the conditions:

$$\text{FREQUENT} \subseteq \{ 2^X \mid X \in \text{MFCS}\}$$
$$\text{INFREQUENT} \subseteq \{ 2^X \mid X \in \text{MFCS}\}$$

Where FREQUENT and INFREQUENT, stand respectively for all frequent and infrequent items (*classified as such as far*).

Pincer-Search Method

Pincer Search combines the following two approaches:

Bottom-up: Generate subsets and go on to generating parent-set candidate sets using frequent subsets.

Top-Down: Generating parent set and then generating subsets.

It also uses two special properties:

Downward Closure Property: If an itemset is frequent, then all its must be frequent.

Upward Closure Property: If an itemset is infrequent, all its supersets must be infrequent.

It uses the above two properties for pruning candidate sets, hence decreases the computation time considerably.

It uses both approaches for pruning in following way:

If some maximal frequent itemset is found in the top down direction, then this itemset can be used to eliminate (possibly many) candidates in the bottom-up direction. The subsets of this frequent itemset will be frequent and hence can be pruned (acc. to Downward closure property).

If an infrequent itemset is found in the bottom up direction, then this infrequent itemset can be used to eliminate the candidates found in top-down search found so far (acc. to Upward Closure Property).

This two-way approach makes use of both the properties and speeds up the search for maximum frequent set.

The algorithm begins with generating 1-itemsets as Apriori algorithm but uses top-down search to prune candidates produced in each pass. This is done with the help of MFCS set.

Let MFS denote set of Maximal Frequent sets storing all maximally frequent itemsets found during the execution. So at anytime during the execution MFCS is a superset of MFS. Algorithm terminates when MFCS equals MFS.

In each pass over database, in addition to counting support counts of candidates in bottom-up direction, the algorithm also counts supports of itemsets in MFCS: this set is adapted for top-down search.

Consider now some pass k, during which itemsets of size k are to be classified. If some itemset that is an element of MFCS, say X of cardinality greater than k is found to be frequent in this pass, then all its subsets will be frequent. Then all its subsets of cardinality k are pruned from the set of candidates considered in bottom-up approach in this pass. They and their supersets will never be candidates again. But, they are not forgotten and used in the end.

Similarly when a new itemsets is found to infrequent in bottom-up direction, the algorithm makes use of it to update MFCS, so that no subset of any itemset in MFCS should have this infrequent itemset as its subset.

By use of MFCS maximal frequent sets can be found early in execution and hence improve performance drastically.

Note: In general, unlike the search in bottom-up direction, which goes up one level in one pass, *the top down search can go down many levels in one pass.*

Now let's see algorithmic representation of Pincer search so that we can see how above concepts are applied:

Pincer Search Method

$L_0 = \emptyset$; k=1; $C_1 = \{\{i\} \mid i \in I\}$; $S_0 = \emptyset$
MFCS = {{ 1,2,…, n}}; MFS = ∅ ;

do until $C_k = \emptyset$ and $S_{k-1} = \emptyset$
    read the database and count support for C_k & MFCS.
MFS = MFS ∪ {frequent itemsets in MFCS};
S_k = {infrequent itemsets in C_k };

call MFCS_gen algorithm if $S_k \neq \emptyset$;
call MFS_pruning procedure;
generate candidates C_{k+1} from C_k ;
if any frequent itemset in C_k is removed from MFS_pruning procedure
call recovery procedure to recover candidates to C_{k+1} .

call MFCS_prune procedure to prune candidates in C_{k+1} .
k=k+1;
return MFS

MFCS_gen
for all itemsets s $\in S_k$
    for all itemsets m $\in$ MFCS
    if s is a subset of m
    MFCS = MFCS \ {m}
    for all items e $\in$ itemset s
        if m \ {e} is not a subset of any itemset in MFCS
        MFCS = MFCS $\cup$ {m \ {e}}
return MFCS

Recovery
for all itemsets l $\in L_k$
    for all itemsets m $\in$ MFS
        if the first k-1 items in l are also in m
            for i from j+1 to |m|
            $C_{k+1} = C_{k+1} \cup \{\{l.item_1, \ldots, l.item_k, m.item_i\}\}$

MFS_prune
for all itemsets c in L_k
if c is a subset of any itemset in the current MFS
delete c from L_k.

MFCS_prune
for all itemsets in C_{k+1}
if c is not a subset of any itemset in the current MFCS
delete c from C_{k+1};

MFCS is initialized to contain one itemset, which contains all of the database items. MFCS is updated whenever new infrequent itemsets are found. If an itemset is found to be frequent then its subsets will not participate in subsequent support counting and candidate generation steps. If some itemsets in L_k are removed, the algorithm will call the *recovery* procedure to recover missing candidates.

Let's apply this algorithm to the following example:

Example :1 - Customer Basket Database

Transaction	Products
1	Burger, Coke, Juice
2	Juice, Potato Chips
3	Coke, Burger
4	Juice, Groundnuts
5	Coke, Groundnuts

Step 1: $L_0 = \varnothing$, k=1;

C_1 = {{ Burger}, {Coke}, {Juice}, {Potato Chips}, {Ground Nuts}}

MFCS = {Burger, Coke, Juice, Potato Chips, Ground Nuts}

MFS = $\varnothing$;

Pass 1: Database is read to count support as follows:

{Burger}→2, {Coke}→3, {Juice}→ 3, {Potato Chips}→1, {Ground Nuts}→2

{Burger, Coke, Juice, Potato Chips, Ground Nuts} → 0

So MFCS = {Burger, Coke, Juice, Potato Chips, Ground Nuts}
& MFS = $\varnothing$;

L_1 = {{Burger}, {Coke}, {Juice}, {Potato Chips}, {Ground Nuts}}
$S_1 = \varnothing$

At the moment since $S_1 = \varnothing$ we don't need to update MFCS

C_2 = {{Burger, Coke}, {Burger, Juice}, {Burger, Potato Chips}, {Burger, Ground Nuts}, {Coke, Juice}, {Coke, Potato Chips}, {Coke, Ground Nuts}, {Juice, Potato Chips}, {Juice, Ground Nuts}, {Potato Chips, Ground Nuts}}

Pass 2: Read database to count support of elements in C_2 & MFCS as given below:

{Burger, Coke} → 2, {Burger, Juice}→1,
{Burger, Potato Chips}→0, {Burger, Ground Nuts}→0,
{Coke, Juice}→1, {Coke, Potato Chips}→0,
{Coke, Ground Nuts}→1, {Juice, Potato Chips}→1,
{Juice, Ground Nuts}→1, {Potato Chips, Ground Nuts}→0

{Burger, Coke, Juice, Potato Chips, Ground Nuts}→0

MFS = ∅

L_2 = {{Burger, Coke}, {Burger, Juice}, {Coke, Juice}, {Coke, Ground Nuts}, {Juice, Ground Nuts}, {Juice, Potato Chips}}

S_2 = {{Burger, Potato Chips}, {Burger, Ground Nuts}, {Coke, Potato Chips}, {Potato Chips, Ground Nuts}}

For {Burger, Potato Chips} in S_2 and for {Burger, Coke, Juice, Potato Chips, Ground Nuts} in MFCS, we get new elements in MFCS as

{Burger, Coke , Juice, Ground Nuts} and {Coke, Juice, Potato Chips, Ground Nuts}

For {Burger, Ground Nuts} in S_2 & for {Coke, Juice, Potato Chips, Ground Nuts} in MFCS, since {Burger, Ground Nuts} is not contained in this element of MFCS hence no action.

For {Burger, Coke, Juice, Ground Nuts} in MFCS we get two new elements
{Burger, Coke, Juice} & {Coke, Juice, Ground Nuts}
Since {Coke, Juice, Ground Nuts} is already contained in MFCS, it is excluded from MFCS

Now, MFCS = {{Burger, Coke, Juice}, {Coke, Juice, Potato Chips, Ground Nuts}}

Now, for {Coke, Potato Chips} in S_2 , we get

MFCS = {{Burger, Coke, Juice}, {Coke, Juice, Ground Nuts}, {Potato Chips, Juice, Ground Nuts}}

Now, for {Potato Chips, Ground Nuts} in S_2, we get

MFCS = {{Burger, Coke, Juice}, {Coke, Juice, Ground Nuts}, {Potato Chips, Juice}}

Now, we generate candidate sets as
C_3 = {{Burger, Coke, Juice}, {Potato Chips}}

Here algorithm stops since no more candidates need to be tested.

Then we do one scan for calculating actual support counts of all maximally frequent itemsets and their subsets. Then we go on to generate rules using minimum confidence threshold to get all *interesting* rules.

x----------x