

# Frequent Itemsets

Hung Le

University of Victoria

January 16, 2019

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $U$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

Support of an itemset  $I$ , denoted by  $\text{Support}(I)$ , is the number of baskets that contains all items in  $I$ .

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $U$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

Support of an itemset  $I$ , denoted by  $\text{Support}(I)$ , is the number of baskets that contains all items in  $I$ .

Baskets	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

$$\text{Support}(\{\text{Bread}, \text{Milk}\}) = 3$$

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $U$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

Baskets	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

If we set  $s = 3$ , then frequent itemsets:

- of size 1 are: {Bread}, {Milk}, {Beer}, {Diapers}.
- of size 2 are: {Bread, Milk}, {Beer, Diapers}

There is no frequent itemset of size at least 3.

# Applications

Discover surprising relationships such as {Beer, Diapers}<sup>1</sup>.

Baskets	Items
1	{Bread,Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread,Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

---

<sup>1</sup>See the history at: <https://tdwi.org/articles/2016/11/15/beer-and-diapers-impossible-correlation.aspx>

# Applications

Detecting plagiarism.

- “Baskets” are sentences.
- “Items” are documents.

Two documents “appear” together in many sentences indicates plagiarism.

# Applications

## Online vs Brick-and-Mortar Retailing

- Online store: if you bought something, they can immediately recommend other items to buy.
- Brick-and-Mortar store: put items in frequent itemsets close to each other on the shelves.

# Association Rule Mining

Rules of the form  $I \rightarrow j$ , often is interpreted as if people buy  $I$ , they will likely buy  $j$  as well.



# Association Rule Mining

Rules of the form  $I \rightarrow j$ , often is interpreted as **if people buy  $I$ , they will likely buy  $j$  as well.**

- Confidence of a rule:

$$\text{Conf}[I \rightarrow j] = \frac{\text{Support}(I \cup \{j\})}{\text{Support}(I)} \quad (1)$$

We typically interested in rules where  $\text{Support}(I) \geq s$ .

- Interest of a rule:

$$\text{Interest}[I \rightarrow j] = \text{Conf}[I \rightarrow j] - \frac{|\text{Support}(\{j\})|}{|\mathcal{B}|} \quad (2)$$

We are typically interested in rules that have positive interest.

# Association Rule Mining

Baskets	Items
1	{ <b>Bread</b> , <b>Milk</b> }
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{ <b>Bread</b> , <b>Milk</b> , Diapers, Beer}
5	{ <b>Bread</b> , <b>Milk</b> , Diapers, Cola}

Association rule:  $\{Beer, Diapers\} \rightarrow Bread$ .

# Association Rule Mining

Baskets	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Association rule:  $\{Beer, Diapers\} \rightarrow Bread$ .

- $\text{Conf}(\{Beer, Diapers\} \rightarrow Bread) = \frac{\text{Support}(\{Beer, Diapers, Bread\})}{\text{Support}(\{Beer, Diapers\})} = \frac{2}{3}$
- $\text{Interest}(\{Beer, Diapers\} \rightarrow Bread) = \frac{2}{3} - \frac{4}{5} < 0$

# Finding Association Rules

Suppose that we are interested in rules that have **confidence at least 0.5**.

## Lemma

*If  $\text{Support}(I) \geq s$  and  $\text{Conf}(I \rightarrow j) \geq 0.5$ , then  $\text{Support}(I \cup \{j\}) \geq s/2$*

# Finding Association Rules

Suppose that we are interested in rules that have **confidence at least 0.5**.

## Lemma

*If  $\text{Support}(I) \geq s$  and  $\text{Conf}(I \rightarrow j) \geq 0.5$ , then  $\text{Support}(I \cup \{j\}) \geq s/2$*

```
FINDASSOCRULES( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $\mathcal{I} \leftarrow \text{FREQUENTITEMSET}(s/2)$   
  for each itemset  $I \in \mathcal{I}$   
     $T[I] \leftarrow \text{Support}(I)$ .    // a hash table  
    for each  $J \in \mathcal{I}$   
      for each  $j \in J$   
         $I \leftarrow J \setminus \{j\}$   
         $c \leftarrow \frac{T[J]}{T[I]}$     // the confidence,  $J = I \cup \{j\}$   
        Report  $I \rightarrow j$  if  $c \geq 0.5$ .
```

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

Some points:

- The number of distinct subsets of  $U$  is  $2^n$  where  $n = |U|$ , so in the worst case the number of “frequent” itemsets is  $2^n$ . (When does this happen?)
- Keep in mind that in reality,  $s$  is set appropriately so that the number of frequent itemsets is not too large. Typically,  $s = 1\%$  to  $10\%$  of the number of baskets.
- Ideally, we would like an algorithm that has running time and memory requirement **linear** to the number of frequent itemsets in the database.

# Frequent Itemset Problem

## Frequent Itemset Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all itemsets  $I$  such that  $\text{Support}(I) \geq s$ .

With very big data, we can't feed all the data to the memory. Thus, we would like an algorithm that:

- passes through the data few times, because reading data from hard disks is very slow.
- minimizes the memory usage.



# A Simplification: Frequent Items

## Frequent Item Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all **items**  $i \in U$  such that **Support** $(\{i\}) \geq s$ .

# A Simplification: Frequent Items

## Frequent Item Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all **items**  $i \in U$  such that **Support** $(\{i\}) \geq s$ .

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
    if Support[ $i$ ]  $\geq s$ .  
      Output  $\{i\}$ .
```

# A Simplification: Frequent Items

## Frequent Item Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $U$  and a threshold  $s$ . Find all **items**  $i \in U$  such that **Support** $(\{i\}) \geq s$ .

```
FREQUENTITEM( $\mathcal{B}, U, s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
    if Support[ $i$ ]  $\geq s$ .  
      Output  $\{i\}$ .
```

Use a hash table or an array (in case all items are indexed from 1 to  $|U|$ ) to implement Support[.].

## A Simplification: Frequent Items

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
      if Support[ $i$ ]  $\geq s$ .  
        Output { $i$ }.
```

- Q: How many passes does the algorithm make?

## A Simplification: Frequent Items

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
      if Support[ $i$ ]  $\geq s$ .  
        Output { $i$ }.
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.

## A Simplification: Frequent Items

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
      if Support[ $i$ ]  $\geq s$ .  
        Output { $i$ }.
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.
- Q: How much memory does the algorithm use?

## A Simplification: Frequent Items

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
      Support[ $i$ ]  $\leftarrow$  Support[ $i$ ] + 1  
      if Support[ $i$ ]  $\geq s$ .  
        Output  $\{i\}$ .
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.
- Q: How much memory does the algorithm use?
  - ▶ Roughly  $32 * n + \text{Size}(U)$  bits where  $n = |U|$ , assuming that a counter of 32 bits suffices to count the frequency of any item, where  $\text{Size}(U)$  is the number of bits in representing items in  $U$ .

# Frequent Itempairs

## Frequent Itempair Problem

Given a set of  $m$  baskets  $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ , each contains a set of items from a ground set  $O$  and a threshold  $s$ . Find all **pairs of elements**  $\{i, j\} \subseteq U$  such that **Support** $(\{i, j\}) \geq s$ .



# Frequent Itempairs

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
      if Support $[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- Q: How many passes does the algorithm make?

# Frequent Itempairs

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
      if Support $[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.

# Frequent Itempairs

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
      if Support $[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.
- Q: How much memory does the algorithm use?

# Frequent Itempairs

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
      if Support $[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- Q: How many passes does the algorithm make?
  - ▶ A: **One** pass.
- Q: How much memory does the algorithm use?
  - ▶ Roughly  $32 * |P| + \text{Size}(P)$  bits where  $P$  is the iset of tempairs that have **non-zero support**.

# Frequent Itempairs : One more pass, fewer memory

## Observation

*If  $\text{Support}(\{i,j\}) \geq s$ , then  $\text{Support}(i) \geq s$  and  $\text{Support}(j) \geq s$ .*

# Frequent Itempairs : One more pass, fewer memory

## Observation

*If  $\text{Support}(\{i, j\}) \geq s$ , then  $\text{Support}(i) \geq s$  and  $\text{Support}(j) \geq s$ .*

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $L_1 \leftarrow \text{FREQUENTITEM}(\mathcal{B}, U, s)$  // a hash table  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$  s.t. both  $i, j \in L_1$   
       $\text{Support}[\{i, j\}] \leftarrow \text{Support}[\{i, j\}] + 1$   
      if  $\text{Support}[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

# Frequent Itempairs : One more pass, fewer memory

## Observation

If  $\text{Support}(\{i, j\}) \geq s$ , then  $\text{Support}(i) \geq s$  and  $\text{Support}(j) \geq s$ .

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $L_1 \leftarrow \text{FREQUENTITEM}(\mathcal{B}, U, s)$  // a hash table  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$  s.t. both  $i, j \in L_1$   
       $\text{Support}[\{i, j\}] \leftarrow \text{Support}[\{i, j\}] + 1$   
      if  $\text{Support}[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- The algorithm makes **two** passes.
- The memory is roughly  $\text{Size}(|L_1|) + |32 * |C_1| + \text{Size}(C_1)$  where  $C_1$  is set of **candidate itempairs**. An itempair is a candidate if **its items are frequent**. We can expect  $C_1 \ll P$ .

# Frequent Itemset of size $k$

## Observation (Monotonicity Principle)

*If  $\text{Support}(I) \geq s$ , then for any subset  $J \subseteq I$ ,  $\text{Support}(J) \geq s$ .*



# Frequent Itemset of size $k$

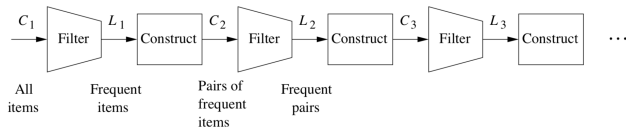
## Observation (Monotonicity Principle)

*If  $\text{Support}(I) \geq s$ , then for any subset  $J \subseteq I$ ,  $\text{Support}(J) \geq s$ .*

```
FREQUENTITEMSET( $\mathcal{B}$ ,  $U$ ,  $s$ ,  $k$ )  
   $L_{k-1} \leftarrow \text{FREQUENTITEMSET}(\mathcal{B}, U, s, k-1)$  // a hash table  
  for each basket  $B$  in  $\mathcal{B}$   
    for each  $k$ -subset  $I \subseteq B$   
      if every  $(k-1)$ -subset  $J$  of  $I$  is in  $L_{k-1}$   
         $\text{Support}[I] \leftarrow \text{Support}[I] + 1$   
        if  $\text{Support}[I] \geq s$   
          Output  $I$ .
```

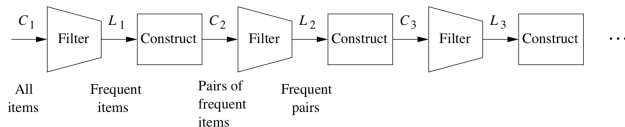
# Frequent Itemset of size $k$

Schematically, the algorithm looks like the following:



# Frequent Itemset of size $k$

Schematically, the algorithm looks like the following:



- The algorithm makes  $k$  passes.
- Memory  $\text{Size}(L_{k-1}) + 32|C_k| + \text{Size}(C_k)$  where  $C_k$  is the set of candidate itemsets of size  $k$ .

# Limited Memory

Recall:

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $L_1 \leftarrow$  FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ ) // a hash table  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$  s.t both  $i, j \in L_1$   
      Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
      if Support $[\{i, j\}] \geq s$ .  
        Output  $\{i, j\}$ .
```

- The memory is roughly  $\text{Size}(L_1) + 32 * |C_1| + \text{Size}(C_1)$  where  $C_1$  is set of **candidate itempairs**. An itempair is a candidate if **its items are frequent**. We want to reduce  $C_1$  further.

# Park-Chen-Yu Algorithm

1st pass:

```
FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each item  $i \in B$   
       $\text{Support}[i] \leftarrow \text{Support}[i] + 1$   
      if  $\text{Support}[i] \geq s$   
        Put  $i$  to  $L_1$   
   $\text{CountMap} \leftarrow \emptyset$  of  $m$  slots  
  for each pair of items  $i, j \in B$   
     $h \leftarrow \text{hash}(i, j)$   
     $\text{CountMap}[h] \leftarrow \text{CountMap}[h] + 1$   
   $\text{BitMap} \leftarrow \emptyset$  of size  $m$   
  for  $h \leftarrow 1$  to  $m$   
    if  $\text{CountMap}[h] \geq s$      $\text{BitMap}[h] \leftarrow 1$   
    else     $\text{BitMap}[h] \leftarrow 0$   
  return  $L_1[.]$ ,  $\text{BitMap}[.]$ 
```

# Park-Chen-Yu Algorithm

2nd pass:

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $L_1[.]$ ,  $BitMap[.] \leftarrow$  FREQUENTITEM( $\mathcal{B}$ ,  $U$ ,  $s$ ) //from 1st pass  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      if both  $i, j \in L_1$  and  $BitMap[hash(\{i, j\})] = 1$   
         $Support[\{i, j\}] \leftarrow Support[\{i, j\}] + 1$   
        if  $Support[\{i, j\}] \geq s$ .  
          Output  $\{i, j\}$ .
```

- The memory is roughly  $Size(L_1) + m + 32 * |C'_1| + Size(C'_1)$  where  $C_1$  is set of **candidate itempairs**. An itempair is a candidate if **its items are frequent and the location of the pair in the BitMap is 1**. Obviously  $C'_1 < C_1$ .
- How can we reduce  $C'_1$  further?

## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$

## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$
- 2nd pass:

CONSTRUCTMAPS( $\mathcal{B}$ ,  $U$ ,  $s$ )

$L_1[.]$ ,  $BitMap_1[.] \leftarrow \text{FREQUENTITEM}(\mathcal{B}, U, s)$  //from 1st pass

$CountMap \leftarrow \emptyset$  of  $m$  slots

**for each** basket  $B$  in  $\mathcal{B}$

**for each** pair of items  $\{i, j\} \in B$

**if** both  $i, j \in L_1$  and  $BitMap_2[hash(\{i, j\})] = 1$

$h \leftarrow hash_2(\{i, j\})$  //a different hash function

$CountMap[h] \leftarrow CountMap[h] + 1$

$BitMap \leftarrow \emptyset$  of size  $m$

**for**  $h \leftarrow 1$  to  $m$

**if**  $CountMap[h] \geq s$      $BitMap_2[h] \leftarrow 1$

**else**     $BitMap_2[h] \leftarrow 0$

return  $L_1[.]$ ,  $BitMap_1[.]$ ,  $BitMap_2[.]$



## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$

## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$
- 2nd pass, construct another bitmap  $BitMap_2[.]$
- 3rd pass:

```
FREQUENTITEMPAIR( $\mathcal{B}$ ,  $U$ ,  $s$ )  
   $L_1[.]$ ,  $BitMap_1[.]$ ,  $BitMap_2[.]$   $\leftarrow$  CONSTRUCTMAPS( $\mathcal{B}$ ,  $U$ ,  $s$ )  
  for each basket  $B$  in  $\mathcal{B}$   
    for each pair of items  $\{i, j\} \in B$   
      if both  $i, j \in L_1$  and  $BitMap_1[hash_1(\{i, j\})] = 1$   
        and  $BitMap_2[hash_2(\{i, j\})] = 1$   
        Support $[\{i, j\}] \leftarrow$  Support $[\{i, j\}] + 1$   
        if Support $[\{i, j\}] \geq s$ .  
          Output  $\{i, j\}$ .
```

# MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$

## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$
- 2nd pass, construct another bitmap  $BitMap_2[.]$
- 3rd pass: check both BitMaps for a candidate pair.

## MultiStage: More passes, more BitMaps, less candidates

- 1st pass: similar to the original PCY. We construct a table of frequent items  $L_1[.]$  and construct the bit map  $BitMap_1[.]$
- 2nd pass, construct another bitmap  $BitMap_2[.]$
- 3rd pass: check both BitMaps for a candidate pair.
- The memory is roughly  $\text{Size}(L_1) + 2m + 32 * |C_1''| + \text{Size}(C_1'')$  where  $C_1$  is set of **candidate itempairs**. An itempair is a candidate if **its items are frequent and the location of the pair in both BitMaps is 1**. Obviously  $C_1'' < C_1'$ .
- How can we reduce  $C_1''$  further? More passes, more BitMaps.

# Limited-Pass Algorithm

What if:

- Your data is too big and I-O is very slow?
- It is OK to find most (but not all) frequent itemsets.

# Limited-Pass Algorithm

Idea: sample a  $p$  fraction of the dataset, say  $p = 1\%$ , and find the frequent itemset in the sample with threshold  $p \cdot s$ . You can expect that your sample, say  $\mathcal{S}$ , can fit into the main memory.

# Limited-Pass Algorithm

Idea: sample a  $p$  fraction of the dataset, say  $p = 1\%$ , and find the frequent itemset in the sample with threshold  $p \cdot s$ . You can expect that your sample, say  $\mathcal{S}$ , can fit into the main memory.

- Frequent itemset in  $\mathcal{S}$  is not frequent in  $\mathcal{B}$ : false positive.
- Frequent itemset in  $\mathcal{B}$  is not frequent in  $\mathcal{S}$ : false negative.



# Limited-Pass Algorithm

Idea: sample a  $p$  fraction of the dataset, say  $p = 1\%$ , and find the frequent itemset in the sample with threshold  $p \cdot s$ . You can expect that your sample, say  $\mathcal{S}$ , can fit into the main memory.

- Frequent itemset in  $\mathcal{S}$  is not frequent in  $\mathcal{B}$ : false positive.
- Frequent itemset in  $\mathcal{B}$  is not frequent in  $\mathcal{S}$ : false negative.

Reducing error:

- Reduce false positive by using one extra pass through the data: count support of all frequent itemsets discovered in the sample.

# Limited-Pass Algorithm

Idea: sample a  $p$  fraction of the dataset, say  $p = 1\%$ , and find the frequent itemset in the sample with threshold  $p \cdot s$ . You can expect that your sample, say  $\mathcal{S}$ , can fit into the main memory.

- Frequent itemset in  $\mathcal{S}$  is not frequent in  $\mathcal{B}$ : false positive.
- Frequent itemset in  $\mathcal{B}$  is not frequent in  $\mathcal{S}$ : false negative.

Reducing error:

- Reduce false positive by using one extra pass through the data: count support of all frequent itemsets discovered in the sample.
- Reduce false negative by lowering the threshold for the sample set: instead of  $p \cdot s$ , we use a smaller threshold, say  $0.9p \cdot s$ .

# Savasere-Omiecinski-Navathe Algorithm

Idea: randomly split the dataset into  $\frac{1}{p}$  disjoint parts, each part of size  $p$  fraction of the whole dataset. A frequent threshold  $p \cdot s$  is set for each part. Find all itemsets that are frequent in **at least one** part.

# Savasere-Omiecinski-Navathe Algorithm

Idea: randomly split the dataset into  $\frac{1}{p}$  disjoint parts, each part of size  $p$  fraction of the whole dataset. A frequent threshold  $p \cdot s$  is set for each part. Find all itemsets that are frequent in **at least one** part.

- No false negative: if the itemset is not frequent in any part, then it is infrequent in the whole dataset. (Why?)

# Savasere-Omiecinski-Navathe Algorithm

Idea: randomly split the dataset into  $\frac{1}{p}$  disjoint parts, each part of size  $p$  fraction of the whole dataset. A frequent threshold  $p \cdot s$  is set for each part. Find all itemsets that are frequent in **at least one** part.

- No false negative: if the itemset is not frequent in any part, then it is infrequent in the whole dataset. (Why?)
- False positive: using one extra pass through the data.