# Frequent Itemsets

Hung Le

University of Victoria

January 15, 2019

# Frequent Itemset Problem

## Frequent Itemset Problem

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

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

# Frequent Itemset Problem

## Frequent Itemset Problem

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

Support of an itemset $I$, denoted by $\texttt{Support}(I)$, 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} |

$\texttt{Support}(\{Brerad, Milk\}) = 3$

# Frequent Itemset Problem

## Frequent Itemset Problem

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

| 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}[1].

| 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} |

[1] 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:

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

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

- Interest of a rule:

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

  We are typically interested in rules that have positive interest.

# 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: $\{Beer, Diapers\} \rightarrow Bread$.

# 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: $\{Beer, Diapers\} \rightarrow Bread$.

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

# Finding Association Rules

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

**Lemma**

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

# Finding Association Rules

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

**Lemma**

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

FindAssocRules($\mathcal{B}$, $U$, $s$)
    $\mathcal{I} \leftarrow$ FrequentItemset($s/2$)
    **for each** itemset $I \in \mathcal{I}$
        $T[I] \leftarrow \texttt{Support}(I)$.  // 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 $I \to j$ if $c \geq 0.5$.

# Frequent Itemset Problem

## Frequent Itemset Problem

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

# Frequent Itemset Problem

## Frequent Itemset Problem

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

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

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

# A Simplification: Frequent Items

## Frequent Item Problem

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

```
FrequentItem(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ s.
                Output {i}.
```

# A Simplification: Frequent Items

## Frequent Item Problem

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

```
FrequentItem(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ 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(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ s.
                Output {i}.
```

- Q: How many passes does the algorithm make?

# A Simplification: Frequent Items

```
FrequentItem(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ s.
                Output {i}.
```

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

# A Simplification: Frequent Items

```
FrequentItem(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ 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(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ s.
                Output {i}.
```

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

# Frequent Itempairs

## Frequent Itempair Problem

Given a set of $m$ baskets $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$, 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 $\texttt{Support}(\{i, j\}) \geq s$.

# Frequent Itempairs

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

- Q: How many passes does the algorithm make?

# Frequent Itempairs

```
FrequentItempair(B, U, s)
    for each basket B in B
        for each pair of items {i, j} ∈ B
            Support[{i, j}] ← Support[{i, j}] + 1
            if Support[{i, j}] ≥ s.
                Output {i, j}.
```

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

# Frequent Itempairs

```
FrequentItempair(B, U, s)
    for each basket B in B
        for each pair of items {i, j} ∈ B
            Support[{i, j}] ← Support[{i, j}] + 1
            if Support[{i, j}] ≥ 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(B, U, s)
    for each basket B in B
        for each pair of items {i, j} ∈ B
            Support[{i, j}] ← Support[{i, j}] + 1
            if Support[{i, j}] ≥ s.
                Output {i, j}.
```

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

# Frequent Itempairs : One more pass, fewer memory

## Observation

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

# Frequent Itempairs : One more pass, fewer memory

## Observation

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

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 $i,j \in L_1$
            $\texttt{Support}[\{i,j\}] \leftarrow \texttt{Support}[\{i,j\}] + 1$
            **if** $\texttt{Support}[\{i,j\}] \geq s$.
                Output $\{i,j\}$.

# Frequent Itempairs : One more pass, fewer memory

## Observation

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

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 $i,j \in L_1$
            $\texttt{Support}[\{i,j\}] \leftarrow \texttt{Support}[\{i,j\}] + 1$
            **if** $\texttt{Support}[\{i,j\}] \geq s$.
                Output $\{i,j\}$.

- The algorithm makes two passes.
- The memory is roughly $\texttt{Size}(|L_1|) + |32 * |C_1| + \texttt{Size}(C_1)$ 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* $\texttt{Support}(I) \geq s$, *then for any subset* $J \subseteq I$, $\texttt{Support}(J) \geq s$.

# Frequent Itemset of size $k$

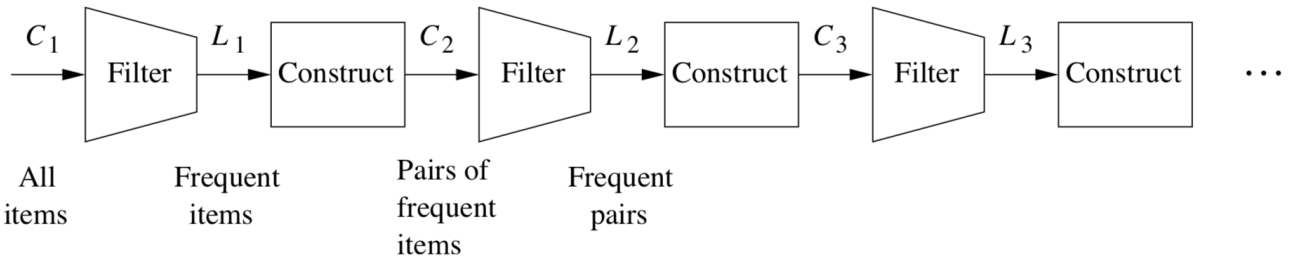
## Observation (Monotonicity Principle)

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

```
FrequentItemset(B, U, s, k)
    L_{k-1} ← FrequentItemset(B, U, s, k − 1) // a hash table
    for each basket B in B
        for each k-subset I ⊆ B
            if every (k − 1)-subset J of I is in L_{k-1}
                Support[I] ← Support[I] + 1
                if Support[I] ≥ s
                    Output I.
```

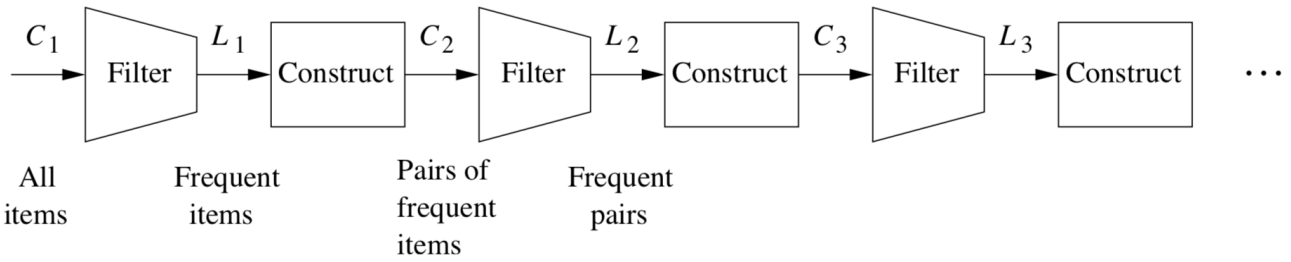
# Frequent Itemset of size $k$

Schematically, the algorithm looks like the following:



# Frequent Itemset of size $k$

Schematically, the algorithm looks like the following:




- The algorithm makes $2k$ passes.
- Memory $\texttt{Size}(L_{k-1}) + 32|C_k| + \texttt{Size}(C_k)$ where $C_k$ is the set of candidate itemsets of size $k$.

# Limited Memory

Recall:

```
FrequentItempair(B, U, s)
    L_1 ← FrequentItem(B, U, s) // a hash table
    for each basket B in B
        for each pair of items {i, j} ∈ B s.t both i, j ∈ L_1
            Support[{i, j}] ← Support[{i, j}] + 1
            if Support[{i, j}] ≥ s.
                Output {i, j}.
```

- The memory is roughly $\texttt{Size}(L_1) + 32 * |C_1| + \texttt{Size}(C_1)$ 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(B, U, s)
    for each basket B in B
        for each item i ∈ B
            Support[i] ← Support[i] + 1
            if Support[i] ≥ s
                Put i to L1
        CountMap ← ∅ of m slots
        for each pair of items i, j ∈ B
            h ← hash(i, j)
            CountMap[h] ← CountMap[h] + 1
    BitMap ← ∅ of size m
    for h ← 1 to m
        if CountMap[h] ≥ s    BitMap[h] ← 1
        else    BitMap[h] ← 0
    return L1[.], 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 $BitMap[hash(\{i, j\})] = 1$
                $\texttt{Support}[\{i, j\}] \leftarrow \texttt{Support}[\{i, j\}] + 1$
                **if** $\texttt{Support}[\{i, j\}] \geq s$.
                    Output $\{i, j\}$.

- The memory is roughly $\texttt{Size}(L_1) + m + 32 * |C_1'| + \texttt{Size}(C_1')$ 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($\mathcal{B}$, $U$, $s$)
    $L_1[.], BitMap_1[.] \leftarrow$ 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($\mathcal{B}$, $U$, $s$)
    $L_1[.], BitMap[.], BitMap_2[.] \leftarrow$ 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 $BitMap_2[hash_2(\{i,j\})] = 1$
                $\texttt{Support}[\{i,j\}] \leftarrow \texttt{Support}[\{i,j\}] + 1$
                **if** $\texttt{Support}[\{i,j\}] \geq s$.
                    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 $\texttt{Size}(L_1) + 2m + 32 * |C_1''| + \texttt{Size}(C_1'')$ 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.