Frequent Itemsets

Hung Le

University of Victoria

February 6, 2019

Misc

- PA1 due by Jan 28, 2019 11:55 pm
- HW1 due by Jan 30, 2019 10:00 am (turn in a hard copy in class).
- For PA1, you don't need to use advanced text processing, i.e, upper case to lower case, remove stop words, remove question mark. Just break a sentence into words by breaking spaces is ok.
- I DO NOT expect you to use complicated subroutine like threading, parallelism to speed up the program.

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 $I \to j$, often is interpreted as if people buy I, they will likely buy j as well.

- Confidence of a rule:

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

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

- Interest of a rule:

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

  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: $\{Beer, Diapers\} \to Bread$.

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

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* $\texttt{Conf}(I \rightarrow j) \geq 0.5$, *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* $\texttt{Support}(I \cup \{j\}) \geq s/2$

$\textsc{FindAssocRules}(\mathcal{B}, U, s)$
    $\mathcal{I} \leftarrow \textsc{FrequentItemset}(s/2)$
    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 U 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 U 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 $\texttt{Support}(\{i\}) \geq s$.

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

A Simplification: Frequent Items

Frequent Item Problem

Given a set of m baskets $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$, 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 $\texttt{Support}[.]$.

A Simplification: Frequent Items

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

- Q: How many passes does the algorithm make?
    - A: One pass.
- Q: How much memory does the algorithm use?
    - Roughly $32 * n + \texttt{Size}(U)$ 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

$\textsc{FrequentItempair}(\mathcal{B}, U, s)$
    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

$\textsc{FrequentItempair}(\mathcal{B}, U, s)$
    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?
  - 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

$\textsc{FrequentItempair}(\mathcal{B}, U, s)$
  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?
  - 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(𝓑, U, s)
    L₁ ← 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 ∈ L₁
            Support[{i,j}] ← Support[{i,j}] + 1
            if Support[{i,j}] ≥ 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(𝓑, U, s)
    L1 ← 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 ∈ L1
            Support[{i,j}] ← Support[{i,j}] + 1
            if Support[{i,j}] ≥ 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

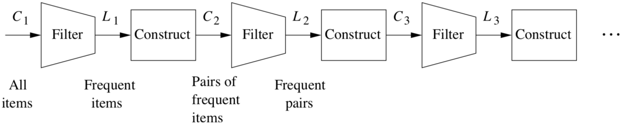
Observation (Monotonicity Principle)

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.
```

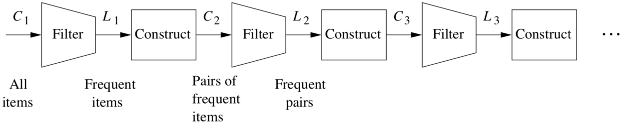
Frequent Itemset of size k

Schematically, the algorithm looks like the following:



Frequent Itemset of size k

Schematically, the algorithm looks like the following:



- The algorithm makes k 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:

$\textsc{FrequentItempair}(\mathcal{B},\ U,\ s)$
  $L_1 \leftarrow \textsc{FrequentItem}(\mathcal{B},\ U,\ s)$ `// 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 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 L_1
        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 L_1[.], BitMap[.]
```

Park-Chen-Yu Algorithm

2nd pass:

```
FrequentItempair(𝓑, U, s)
    L_1[.], BitMap[.] ← FrequentItem(𝓑, U, s) //from 1st pass
    for each basket B in 𝓑
        for each pair of items {i, j} ∈ B
            if both i, j ∈ L_1 and BitMap[hash({i, j})] = 1
                Support[{i, j}] ← Support[{i, j}] + 1
                if Support[{i, j}] ≥ 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(𝓑, U, s)
    L₁[.], BitMap₁[.] ← FrequentItem(𝓑, U, s) //from 1st pass
    CountMap ← ∅ of m slots
    for each basket B in 𝓑
        for each pair of items {i, j} ∈ B
            if both i, j ∈ L₁ and BitMap₁[hash({i, j})] = 1
                h ← hash₂({i, j}) //a different hash function
                CountMap[h] ← CountMap[h] + 1
    BitMap₂ ← ∅ of size m
    for h ← 1 to m
        if CountMap[h] ≥ s     BitMap₂[h] ← 1
        else    BitMap₂[h] ← 0
    return L₁[.], BitMap₁[.], BitMap₂[.]
```

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(𝓑, U, s)
    L₁[.], BitMap[.], BitMap₂[.] ← ConstructMaps(𝓑, U, s)
    for each basket B in 𝓑
        for each pair of items {i, j} ∈ B
            if both i, j ∈ L₁ and BitMap₁[hash₁({i, j})] = 1
                    and BitMap₂[hash₂({i, j})] = 1
                Support[{i, j}] ← Support[{i, j}] + 1
                if Support[{i, j}] ≥ 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.

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 lowing 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.