Machine Learning for Data Mining

Frequent Itemset Mining & Association Rules

Andres Mendez-Vazquez

August 25, 2016

Outline

Outline

Association Rule Discovery

In the Market-basket model

Goal: **Identify items that are bought together by enough customers to be significant.**

Association Rule Discovery

In the Market-basket model

Goal: **Identify items that are bought together by enough customers to be significant.**

What do we can do?

Process the collected sales data using the barcode ID to find dependencies among items.

- Thus, do not be surprised if you find six packs next to diapers!!!

Association Rule Discovery

In the Market-basket model

Goal: **Identify items that are bought together by enough customers to be significant.**

What do we can do?

Process the collected sales data using the barcode ID to find dependencies among items.

We can use the following classic observation

- If one buys diaper and milk, then he is likely to buy beer!!!
- Thus, do not be surprised if you find six packs next to diapers!!!

Association Rule Discovery

In the Market-basket model

Goal: **Identify items that are bought together by enough customers to be significant.**

What do we can do?

Process the collected sales data using the barcode ID to find dependencies among items.

We can use the following classic observation

- If one buys diaper and milk, then he is likely to buy beer!!!
- Thus, do not be surprised if you find six packs next to diapers!!!

The Market-Basket Model

A large set of items

For example, things sold in a supermarket.

A large set of baskets, which is a small subset of items

For example, the things one customer buys on one day.

In general, we have a many to many mapping (association) between two types of things

However, we are asking about connections among "items", not "baskets."

The Market-Basket Model

A large set of items

For example, things sold in a supermarket.

A large set of baskets, which is a small subset of items

For example, the things one customer buys on one day.

In general, we have a many to many mapping (association) between two types of things

However, we are asking about connections among "items", not "baskets."

The Market-Basket Model

A large set of items

For example, things sold in a supermarket.

A large set of baskets, which is a small subset of items

For example, the things one customer buys on one day.

In general, we have a many to many mapping (association) between two types of things

However, we are asking about connections among "items", not "baskets."

Outline

Association Rules

Given a set of baskets

ID	Items
1	Bread,Coke,Milk
2	Beer,Bread
3	Beer,Coke,Diaper,Milk
4	Beer,Bread,Diaper,Milk
5	Coke,Diaper,Milk

We want to discover association rules

- People who bought $\{x, y, z\}$ tend to buy $\{v, w\}$.

Association Rules

Given a set of baskets

ID	Items
1	Bread,Coke,Milk
2	Beer,Bread
3	Beer,Coke,Diaper,Milk
4	Beer,Bread,Diaper,Milk
5	Coke,Diaper,Milk

We want to discover association rules

- People who bought $\{x, y, z\}$ tend to buy $\{v, w\}$.

Itemsets

Basically

Given the baskets we want to find if an itemset (**Set of items**) is a likely set.

Association Rules

And given that

We want to generate likely association rules

Output:

Rules Discovered
{Milk}$\Rightarrow${Coke}
{Diaper,Milk}$\Rightarrow${Beer}

Association Rules

And given that

We want to generate likely association rules

Output:

Rules Discovered
{Milk}⇒{Coke}
{Diaper,Milk}⇒{Beer}

Outline

Applications: Market Analysis

Items and Baskets

- Items are products at the store.
- Baskets are sets of products someone bought in one trip to the store.

Applications: Market Analysis

Items and Baskets

- Items are products at the store.
- Baskets are sets of products someone bought in one trip to the store.

Real market baskets

Chain stores keep Tera-bytes of data about what customers buy together

- It tells them how customers navigate stores, thus allowing them position tempting items

It suggests "marketing tricks", for example, run sales on diapers and raise the price of beer

- Nevertheless, This needs High Support (A lot of Data), or no Money!!!

Real market baskets

Chain stores keep Tera-bytes of data about what customers buy together

- It tells them how customers navigate stores, thus allowing them position tempting items

It suggests "marketing tricks", for example, run sales on diapers and raise the price of beer

- Nevertheless, This needs High Support (A lot of Data), or no Money!!!

Applications

Baskets = sentences; Items = documents containing those sentences

Items that appear together too often could represent plagiarism

Applications

Baskets = sentences; Items = documents containing those sentences

Items that appear together too often could represent plagiarism

Baskets = patients; Items = drugs and side-effects

- It has been used to detect combinations of drugs that result in particular side-effects
- However, it requires an extension: Absence of an item needs to be observed as well as its presence

Applications

Baskets = sentences; Items = documents containing those sentences

Items that appear together too often could represent plagiarism

Baskets = patients; Items = drugs and side-effects

- It has been used to detect combinations of drugs that result in particular side-effects
- However, it requires an extension: Absence of an item needs to be observed as well as its presence

Applications - Finding communities in graphs (e.g. the Web)

If we are looking for communities

It is possible to use the idea of clique to find a community in a graph!!!

Applications - Finding communities in graphs (e.g. the Web)

If we are looking for communities

It is possible to use the idea of clique to find a community in a graph!!!

Problem

This is a complete NP-complete problem.

We avoid this problem by using the following trick

Given a graph

- Divide the nodes into two equal groups at random.

- We expect that about half of its edges would go between groups.

We avoid this problem by using the following trick

Given a graph

- Divide the nodes into two equal groups at random.

If a community exist by defining "Between each two nodes exist an edge"

- We expect that about half of its nodes to fall into each group.
- We expect that about half of its edges would go between groups.

We avoid this problem by using the following trick

Given a graph

- Divide the nodes into two equal groups at random.

If a community exist by defining "Between each two nodes exist an edge"

- We expect that about half of its nodes to fall into each group.
- We expect that about half of its edges would go between groups.

Baskets = Nodes in the Left and Items = Nodes in the Right

The problem becomes on a search of complete bipartite subgraphs $K_{s,t}$ on a Bipartite Graph

- Thus, given a community kernel representing it, we add nodes from either of the two groups.

By Using a Simple Rule

- if those nodes have edges to many of the nodes already identified as belonging to the community.

Baskets = Nodes in the Left and Items = Nodes in the Right

The problem becomes on a search of complete bipartite subgraphs $K_{s,t}$ on a Bipartite Graph

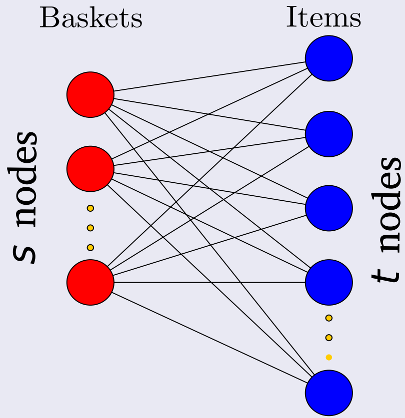
- Thus, given a community kernel representing it, we add nodes from either of the two groups.

By Using a Simple Rule

- if those nodes have edges to many of the nodes already identified as belonging to the community.

Applications - Finding communities in graphs (e.g. the Web)

For Example



Applications - Finding communities in graphs (e.g. the Web)

How?

The members of the basket, for node v, are the nodes of the left side to which v is connected.

Let the support threshold be s

The number of nodes that the instance of $K_{s,t}$ has on the right side.

Looking for $K_{s,t}$ is like looking for a set of support s with a layer t

Or, all frequent itemsets of size t.

Applications - Finding communities in graphs (e.g. the Web)

How?

The members of the basket, for node v, are the nodes of the left side to which v is connected.

Let the support threshold be s

The number of nodes that the instance of $K_{s,t}$ has on the right side.

Looking for $K_{s,t}$ is like looking for a set of support s with a layer t

Or, all frequent itemsets of size t

Applications - Finding communities in graphs (e.g. the Web)

How?

The members of the basket, for node v, are the nodes of the left side to which v is connected.

Let the support threshold be s

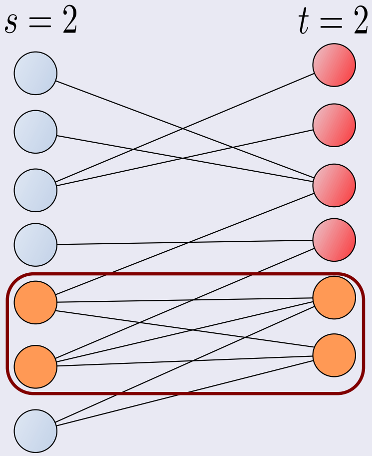
The number of nodes that the instance of $K_{s,t}$ has on the right side.

Looking for $K_{s,t}$ is like looking for a set of support s with a layer t

Or, all frequent itemsets of size t

That is

If a set of t nodes on the right side is frequent, then they all occur together in at least s baskets



Outline

The Basics

The set of all items in a market basket data is defined as

$$\mathcal{I} = \{i_1, i_2, ..., i_d\} \quad (1)$$

The set of all transactions (Baskets)

$$\mathcal{T} = \{t_1, t_2, ..., t_N\} \quad (2)$$

Where

Each transaction t_i contains subsets of items chosen from $\mathcal{I}$.

The Basics

The set of all items in a market basket data is defined as

$$\mathcal{I} = \{i_1, i_2, ..., i_d\} \tag{1}$$

The set of all transactions (Baskets)

$$\mathcal{T} = \{t_1, t_2, ..., t_N\} \tag{2}$$

Where

Each transaction t_i contains subsets of items chosen from $\mathcal{I}$.

The Basics

The set of all items in a market basket data is defined as

$$\mathcal{I} = \{i_1, i_2, ..., i_d\} \quad (1)$$

The set of all transactions (Baskets)

$$\mathcal{T} = \{t_1, t_2, ..., t_N\} \quad (2)$$

Where

Each transaction t_i contains subsets of items chosen from $\mathcal{I}$.

Itemsets

Defintion

An itemset is any of the subsets from $\mathcal{I}$.

Thus

A transaction t_i is said to contain an Itemset I, if I is a subset of t_i.

Itemsets

Defintion

An itemset is any of the subsets from $\mathcal{I}$.

Thus

A transaction t_i is said to contain an Itemset I, if I is a subset of t_i.

Support of an Itemset

We define the support for itemset I as

- Number of baskets containing all items in I
  - Often expressed as a fraction of the total number of baskets.

Support of an Itemset

We define the support for itemset I as

- Number of baskets containing all items in I
  - Often expressed as a fraction of the total number of baskets.

Definition

$$\sigma(I) = |\{t_i | I \subseteq t_i, t_i \in T\}| \quad (3)$$

Support of an Itemset

We define the support for itemset I as

- Number of baskets containing all items in I
  - Often expressed as a fraction of the total number of baskets.

Definition

$$\sigma(I) = |\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}| \tag{3}$$

Then

Given a support threshold s, then sets that appear in at least s baskets are called frequent itemsets.

Support of an Itemset

We define the support for itemset I as

- Number of baskets containing all items in I
  - Often expressed as a fraction of the total number of baskets.

Definition

$$\sigma(I) = |\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}| \tag{3}$$

Then

Given a **support threshold** s, then sets that appear in at least s baskets are called frequent itemsets

Question

Now, we ask a really simplest question

Can you find sets of items that appear together "**frequently**" in the baskets?

Example of Frequent Itemsets

Items

Given a set $X = \{milk, coke, pepsi, beer, juice\}$

And the following baskets, we are looking the itemsets with support $s = 3$

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

Thus, the Frequent Itemsets

$\{m\}, \{c\}, \{b\}, \{j\}, \{m, b\}, \{b, c\}, \{c, j\}$.

Example of Frequent Itemsets

Items

Given a set $X = \{milk, coke, pepsi, beer, juice\}$

And the following baskets, we are looking the itemsets with support $s = 3$

$$
\begin{array}{llll}
B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\
B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\}
\end{array}
$$

Thus, the Frequent Itemsets

$\{m\}, \{c\}, \{b\}, \{j\}, \{m, b\}, \{b, c\}, \{c, j\}.$

Example of Frequent Itemsets

Items

Given a set $X = \{milk, coke, pepsi, beer, juice\}$

And the following baskets, we are looking the itemsets with support $s = 3$

$$
\begin{array}{llll}
B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\
B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\}
\end{array}
$$

Thus, the Frequent Itemsets

$\{m\}, \{c\}, \{b\}, \{j\}, \{m, b\}, \{b, c\}, \{c, j\}$.

Problem

We have $2^{|X|} - 1$ sets to explore

Can we do better?

How do we deal with this?

Using the Apriori Property

Problem

We have $2^{|X|} - 1$ sets to explore

Can we do better?

How do we deal with this?

Using the Apriori Property

Apriory Principle

Theorem (Apriori Principle)

If an itemset is frquent, then also all of its subset must also be frequent.

Apriory Principle

Theorem (Apriori Principle)

If an itemset is frquent, then also all of its subset must also be frequent.

The idea is based on the following observations

1. If an itemset I does not satisfy the minimum support threshold, i.e. $support(I) < s \Rightarrow I$ is not frequent.
2. If an item A is added to the itemset I i.e. $\{A\} \cup I$, then the resulting itemset cannot occur more frequently than I.
   - Thus, $I \cup A$ is not frequent or $\sigma(I \cup A) < s$.

Apriory Principle

Theorem (Apriori Principle)

If an itemset is frquent, then also all of its subset must also be frequent.

The idea is based on the following observations

1. If an itemset I does not satisfy the minimum support threshold, i.e. $support\,(I) < s \Rightarrow I$ is not frequent.
2. **If an item A is added to the itemset I i.e. $\{A\} \cup I$, then the resulting itemset cannot occur more frequently than I.**
    - Thus, $I \cup A$ is not frequent or $\sigma\,(I \cup A) < s$.

Proof

First, we prove that if itemset I is frequent then the subset are frequent

Given a transaction t_i, such that $I \subseteq t_i$, then for any subset $A \subseteq I \longrightarrow A \subseteq t_i$. Now as a result that $\sigma(I) \geq s$.

We can use the Monotonicity Property

Let I be a set of items, and $J = 2^I$ be the power set of I. A measure f is monotone if

$$\forall X, Y \in J \text{ if } X \subseteq Y \longrightarrow f(X) \leq f(Y) \quad (4)$$

Clearly

The cardinality is a monotone measure.

Proof

First, we prove that if itemset I is frequent then the subset are frequent

Given a transaction t_i, such that $I \subseteq t_i$, then for any subset $A \subseteq I \longrightarrow A \subseteq t_i$. Now as a result that $\sigma(I) \geq s$.

We can use the Monotonicity Property

Let I be a set of items, and $J = 2^I$ be the power set of I. A measure f is monotone if

$$\forall X, Y \in J \text{ if } X \subseteq Y \longrightarrow f(X) \leq f(Y) \quad (4)$$

Clearly

The cardinality is a monotone measure.

Proof

First, we prove that if itemset I is frequent then the subset are frequent

Given a transaction t_i, such that $I \subseteq t_i$, then for any subset $A \subseteq I \longrightarrow A \subseteq t_i$. Now as a result that $\sigma(I) \geq s$.

We can use the Monotonicity Property

Let I be a set of items, and $J = 2^I$ be the power set of I. A measure f is monotone if

$$\forall X, Y \in J \text{ if } X \subseteq Y \longrightarrow f(X) \leq f(Y) \quad (4)$$

Clearly

The cardinality is a monotone measure.

Proof

Thus, given that $\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I \subseteq \{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A$

$$|\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I| \leq |\{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A| \quad (5)$$

Or

$$s < \sigma(I) \leq \sigma(A) \quad (6)$$

The itemset A is frequent.

Now assume that an itemset A is infrequent and there is a superset I i.e. $A \subseteq I$

Then, given that $\sigma(A) < s$ and $|\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I| \leq |\{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A|$ then $\sigma(I) \leq \sigma(A) < s$ i.e. I is infrequent

Q.E.D.

Proof

Thus, given that $\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I \subseteq \{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A$

$$\left|\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I\right| \leq \left|\{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A\right| \tag{5}$$

Or

$$s < \sigma(I) \leq \sigma(A) \tag{6}$$

The itemset A is frequent.

Now assume that an itemset A is infrequent and there is a superset I i.e. $A \subseteq I$

Then, given that $\sigma(A) < s$ and $\left|\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}_I\right| \leq \left|\{t_i | A \subseteq t_i, t_i \in \mathcal{T}\}_A\right|$ then $\sigma(I) \leq \sigma(A) < s$ i.e. I is infrequent

Q.E.D.

Proof

Thus, given that $\{t_i|I \subseteq t_i, t_i \in \mathcal{T}\}_I \subseteq \{t_i|A \subseteq t_i, t_i \in \mathcal{T}\}_A$

$$|\{t_i|I \subseteq t_i, t_i \in \mathcal{T}\}_I| \leq |\{t_i|A \subseteq t_i, t_i \in \mathcal{T}\}_A| \quad (5)$$

Or

$$s < \sigma(I) \leq \sigma(A) \quad (6)$$

The itemset A is frequent.

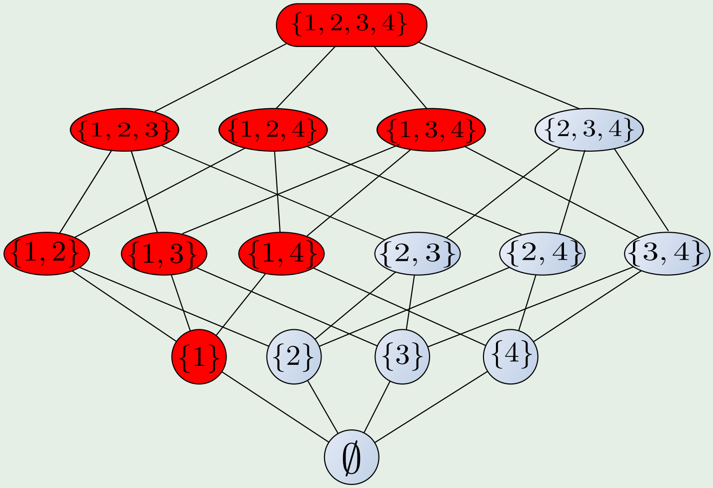
Now assume that an itemset A is infrequent and there is a superset I i.e. $A \subseteq I$

Then, given that $\sigma(A) < s$ and $|\{t_i|I \subseteq t_i, t_i \in \mathcal{T}\}_I| \leq |\{t_i|A \subseteq t_i, t_i \in \mathcal{T}\}_A|$ then $\sigma(I) \leq \sigma(A) < s$ i.e. I is infrequent

Q.E.D.

This principle allows to prune the power set

Example for $\{1\}$ not frequent



Outline

Now, Back To The Association Rules

Association Rules

They are If-then rules about the contents of baskets.

Now, Back To The Association Rules

Association Rules

They are If-then rules about the contents of baskets.

Definition

- $I \rightarrow \{j\}$ means: "if a basket contains all of $I = \{i_1, ..., i_k\}$ then it is likely to contain j"
- In practice there are many rules, we want to find significant/interesting ones!
- Thus, we can define the concept of **Confidence** for rule $(I \rightarrow \{j\})$ as the sampling probability of j given I

Now, Back To The Association Rules

Association Rules

They are If-then rules about the contents of baskets.

Definition

- $I \rightarrow \{j\}$ means: "if a basket contains all of $I = \{i_1, ..., i_k\}$ then it is likely to contain j"
- In practice there are many rules, we want to find significant/interesting ones!
- Thus, we can define the concept of **Confidence** for rule $(I \rightarrow \{j\})$ as the sampling probability of j given I

The the **Confidence** is given by

$$conf(I \rightarrow \{j\}) = \frac{\sigma(I \cup \{j\})}{\sigma(I)}$$

Now, Back To The Association Rules

Association Rules

They are If-then rules about the contents of baskets.

Definition

- $I \to \{j\}$ means: "if a basket contains all of $I = \{i_1, ..., i_k\}$ then it is likely to contain j"
- In practice there are many rules, we want to find significant/interesting ones!
- Thus, we can define the concept of **Confidence** for rule $(I \to \{j\})$ as the sampling probability of j given I

The the **Confidence** is given by

$$conf(I \to \{j\}) = \frac{\sigma(I \cup \{j\})}{\sigma(I)}$$

cinvestav

Now, Back To The Association Rules

Association Rules

They are If-then rules about the contents of baskets.

Definition

- $I \rightarrow \{j\}$ means: "if a basket contains all of $I = \{i_1, ..., i_k\}$ then it is likely to contain j"
- In practice there are many rules, we want to find significant/interesting ones!
- Thus, we can define the concept of **Confidence** for rule $(I \rightarrow \{j\})$ as the sampling probability of j given I

The the **Confidence** is given by

$$conf(I \rightarrow \{j\}) = \frac{\sigma(I \cup \{j\})}{\sigma(I)}$$

However

Not all high-confidence rules are interesting

- It is possible to have high confidence for many itemsets I without creating interesting rules.
- For example, milk is just purchased very often (independent of I) making the confidence high,
  - but not all the rules based on milk are interesting.

However

Not all high-confidence rules are interesting

- It is possible to have high confidence for many itemsets I without creating interesting rules.
- For example, milk is just purchased very often (independent of I) making the confidence high,
  - but not all the rules based on milk are interesting.

However

Not all high-confidence rules are interesting

- It is possible to have high confidence for many itemsets I without creating interesting rules.
- For example, milk is just purchased very often (independent of I) making the confidence high,
  - but not all the rules based on milk are interesting.

Defining Interest

Thus

We can define a better measure to find interesting rules.

Defining Interest

Thus

We can define a better measure to find interesting rules.

Definition

The interest function is the difference between its confidence and the fraction of baskets that contain j

$$Interest(I \rightarrow \{j\}) = conf(I \rightarrow j) - Pr(\{j\})$$

Where

$$Pr(\{j\}) = \frac{|\{t_i | I \subseteq t_i, t_i \in T\}|}{\text{Numer of Baskets}} \quad (7)$$

Defining Interest

Thus

We can define a better measure to find interesting rules.

Definition

The interest function is the difference between its confidence and the fraction of baskets that contain j

$$Interest(I \rightarrow \{j\}) = conf(I \rightarrow j) - Pr(\{j\})$$

Where

$$Pr(\{j\}) = \frac{|\{t_i | I \subseteq t_i, t_i \in \mathcal{T}\}|}{\text{Numer of Baskets}} \quad (7)$$

Interesting Association Rules

Interesting rules are those with high positive or negative interest values

For this, we have that

$$Pr[j] \gg conf\,(I \to j) \text{ or } conf\,(I \to j) \gg Pr[j] \quad (8)$$

Making the interest low

Interesting Association Rules

Interesting rules are those with high positive or negative interest values

For this, we have that

$$Pr[j] \gg conf\left(I \rightarrow j\right) \text{ or } conf\left(I \rightarrow j\right) \gg Pr[j] \quad (8)$$

For the uninteresting rules, we have that

- The fraction of baskets containing j will be the same as the fraction of the subset baskets including $\{I, j\}$
- Making the interest low.

Interesting Association Rules

Interesting rules are those with high positive or negative interest values

For this, we have that

$$Pr[j] \gg conf\left(I \rightarrow j\right) \text{ or } conf\left(I \rightarrow j\right) \gg Pr[j] \tag{8}$$

For the uninteresting rules, we have that

- The fraction of baskets containing j will be the same as the fraction of the subset baskets including $\{I, j\}$
- Making the interest low.

Interesting Association Rules

Interesting rules are those with high positive or negative interest values

For this, we have that

$$Pr[j] \gg conf\left(I \rightarrow j\right) \text{ or } conf\left(I \rightarrow j\right) \gg Pr[j] \quad (8)$$

For the uninteresting rules, we have that

- The fraction of baskets containing j will be the same as the fraction of the subset baskets including $\{I, j\}$
- Making the interest low.

Example of Confidence and Interest

Given the following collection of baskets

$$\begin{array}{llll} B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\ B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\} \end{array}$$

Example of Confidence and Interest

Given the following collection of baskets

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

We measure the association rule $\{m, b\} \rightarrow c$

Thus, we have that

- Confidence $= 2/4 = 0.5$
- Interest $= 0.5 – 5/8 = -1/8$
  - Item c appears in $5/8$ of the baskets
  - **Thus, the rule is not very interesting!**

Example of Confidence and Interest

Given the following collection of baskets

$$
\begin{array}{llll}
B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\
B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\}
\end{array}
$$

We measure the association rule $\{m, b\} \rightarrow c$

Thus, we have that

- Confidence $= 2/4 = 0.5$
- Interest $= 0.5 – 5/8 = -1/8$
    - Item c appears in $5/8$ of the baskets
    - **Thus, the rule is not very interesting!**

Example of Confidence and Interest

Given the following collection of baskets

$$\begin{array}{llll} B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\ B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\} \end{array}$$

We measure the association rule $\{m, b\} \rightarrow c$

Thus, we have that

- Confidence $= 2/4 = 0.5$
- Interest $= 0.5 – 5/8 = -1/8$
    - Item c appears in $5/8$ of the baskets
    - Thus, the rule is not very interesting!

Example of Confidence and Interest

Given the following collection of baskets

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

We measure the association rule $\{m, b\} \rightarrow c$

Thus, we have that

- Confidence $= 2/4 = 0.5$
- Interest $= 0.5 – 5/8 = -1/8$
  - Item c appears in $5/8$ of the baskets
  - Thus, the rule is not very interesting!

Example of Confidence and Interest

Given the following collection of baskets

$$
\begin{array}{llll}
B_1 = \{m, c, b\} & B_2 = \{m, p, j\} & B_3 = \{m, b\} & B_4 = \{c, j\} \\
B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} & B_7 = \{c, b, j\} & B_8 = \{b, c\}
\end{array}
$$

We measure the association rule $\{m, b\} \rightarrow c$

Thus, we have that

- Confidence $= 2/4 = 0.5$
- Interest $= 0.5–5/8 = -1/8$
  - Item c appears in $5/8$ of the baskets
  - **Thus, the rule is not very interesting!**

Finding Association Rules

Problem

Find all association rules with support$\geq s$ and confidence$\geq c$

Where the **support** of an association rules is defined as

$$s\left(I \rightarrow \{j\}\right) = \frac{\sigma\left(I \cup \{j\}\right)}{\text{Numer of Baskets}} = \frac{\sigma\left(I \cup \{j\}\right)}{N} \quad (9)$$

Finding Association Rules

Problem

Find all association rules with support$\geq s$ and confidence$\geq c$

Where the support of an association rules is defined as

$$s\left(I \rightarrow \{j\}\right) = \frac{\sigma\left(I \cup \{j\}\right)}{\text{Numer of Baskets}} = \frac{\sigma\left(I \cup \{j\}\right)}{N} \tag{9}$$

Finding Association Rules

The Hard part!!! Finding the frequent itemsets!!!

If $I \rightarrow \{j\}$ has high support and confidence, then both I and $I \cup \{j\}$ will be "frequent"

Again

$$conf\left(I \rightarrow \{j\}\right) = \frac{\sigma\left(I \cup \{j\}\right)}{\sigma\left(I\right)}$$

Finding Association Rules

The Hard part!!! Finding the frequent itemsets!!!

If $I \rightarrow \{j\}$ has high support and confidence, then both I and $I \cup \{j\}$ will be "frequent"

Again

$$conf\left(I \rightarrow \{j\}\right) = \frac{\sigma\left(I \cup \{j\}\right)}{\sigma\left(I\right)}$$

Important Observation

First

Often, small frequent itemsets are quite more "frequent"

- to the point that k never grows beyond 2 or 3.

Important Observation

First

Often, small frequent itemsets are quite more “frequent”

- to the point that k never grows beyond 2 or 3.

Second

- When looking for itemsets for a large size k.
- It is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset.
- Thus, the value of n drops as k increases.

Important Observation

First

Often, small frequent itemsets are quite more "frequent"

- to the point that k never grows beyond 2 or 3.

Second

- When looking for itemsets for a large size k.
- It is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset.
- Thus, the value of n drops as k increases.

Important Observation

First

Often, small frequent itemsets are quite more "frequent"

- to the point that k never grows beyond 2 or 3.

Second

- When looking for itemsets for a large size k.
- It is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset.
- Thus, the value of n drops as k increases.

Important Observation

First

Often, small frequent itemsets are quite more “frequent”

- to the point that k never grows beyond 2 or 3.

Second

- When looking for itemsets for a large size k.
- It is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset.
- Thus, the value of n drops as k increases.

Outline

Association Rules Process

Step 1: Find all frequent itemsets I

We will explain this later in the presentation.

Association Rules Process

Step 1: Find all frequent itemsets I

We will explain this later in the presentation.

Step 2: Rule generation

- For every subset A of I, generate a rule $A \rightarrow I - A$
  - Since I is frequent, A is also frequent
  - Calculate the confidences
- Output the rules above the confidence threshold ϵ.

Association Rules Process

Step 1: Find all frequent itemsets I

We will explain this later in the presentation.

Step 2: Rule generation

- For every subset A of I, generate a rule $A \to I - A$
  - Since I is frequent, A is also frequent
  - Calculate the confidences
- Output the rules above the confidence threshold ϵ.

Association Rules Process

Step 1: Find all frequent itemsets I

We will explain this later in the presentation.

Step 2: Rule generation

- For every subset A of I, generate a rule $A \rightarrow I - A$
  - Since I is frequent, A is also frequent
  - Calculate the confidences
- Output the rules above the confidence threshold ϵ.

Association Rules Process

Step 1: Find all frequent itemsets I

We will explain this later in the presentation.

Step 2: Rule generation

- For every subset A of I, generate a rule $A \rightarrow I - A$
  - Since I is frequent, A is also frequent
  - Calculate the confidences
- Output the rules above the confidence threshold ϵ.

We have two variants for calculating to confidence

Variant 1

Single pass to compute the rule of confidence:

$$conf\left(\{A,B\}\rightarrow\{C,D\}\right)=\frac{\sigma\left(\{A,B,C,D\}\right)}{\sigma\left(\{A,B\}\right)} \quad (10)$$

We have two variants for calculating to confidence

Variant 1

Single pass to compute the rule of confidence:

$$conf\left(\{A,B\}\rightarrow\{C,D\}\right)=\frac{\sigma\left(\{A,B,C,D\}\right)}{\sigma\left(\{A,B\}\right)} \tag{10}$$

Variant 2

- **Observation**:
  - If $\{A,B,C\}\rightarrow\{D\}$ is below confidence, so is $\{A,B\}\rightarrow\{C,D\}$
- Thus It possible to generate "bigger" rules (More items in the antecedent and consequent) from smaller ones,
  - If they are above confidence!!!

We have two variants for calculating to confidence

Variant 1

Single pass to compute the rule of confidence:

$$conf\left(\{A,B\}\rightarrow\{C,D\}\right)=\frac{\sigma\left(\{A,B,C,D\}\right)}{\sigma\left(\{A,B\}\right)} \tag{10}$$

Variant 2

- **Observation**:
  - If $\{A,B,C\}\rightarrow\{D\}$ is below confidence, so is $\{A,B\}\rightarrow\{C,D\}$
- Thus It possible to generate "bigger" rules (More items in the antecedent and consequent) from smaller ones,
  - If they are above confidence!!!

We have two variants for calculating to confidence

Variant 1

Single pass to compute the rule of confidence:

$$conf\left(\{A, B\} \to \{C, D\}\right) = \frac{\sigma\left(\{A, B, C, D\}\right)}{\sigma\left(\{A, B\}\right)} \tag{10}$$

Variant 2

- **Observation**:
  - If $\{A, B, C\} \to \{D\}$ is below confidence, so is $\{A, B\} \to \{C, D\}$
- Thus It possible to generate "bigger" rules (More items in the antecedent and consequent) from smaller ones,
  - If they are above confidence!!!

We have two variants for calculating to confidence

Variant 1

Single pass to compute the rule of confidence:

$$conf\left(\{A,B\}\rightarrow\{C,D\}\right)=\frac{\sigma\left(\{A,B,C,D\}\right)}{\sigma\left(\{A,B\}\right)} \tag{10}$$

Variant 2

- **Observation**:
  - If $\{A,B,C\}\rightarrow\{D\}$ is below confidence, so is $\{A,B\}\rightarrow\{C,D\}$
- Thus It possible to generate "bigger" rules (More items in the antecedent and consequent) from smaller ones,
  - If they are above confidence!!!

Example

We have a bunch of baskets

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

- We have a minimum support $s = 3$ with confidence $c = 0.75$

Frequent itemsets

$\{b, m\} \{b, c\}$ $\{c, m\}$ $\{c, j\}$ $\{m, c, b\}$

Generate rules by eliminating anything below $c = 0.75$

Rule	Confidence	Remove	Rule	Confidence	Remove
$b \to m$	$c = 4/6$	Yes	$b, c \to m$	$c = 3/5$	Yes
$m \to b$	$c = 4/5$	No	$b, m \to c$	$c = 3/4$	No
	$\vdots$			$\vdots$	

Example

We have a bunch of baskets

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

- We have a minimum support $s = 3$ with confidence $c = 0.75$

Frequent itemsets

$\{b, m\}\{b, c\}$ $\{c, m\}$ $\{c, j\}$ $\{m, c, b\}$

Generate rules by eliminating anything below $c = 0.75$

Rule	Confidence	Remove	Rule	Confidence	Remove
$b \to m$	$c = 4/6$	Yes	$b, c \to m$	$c = 3/5$	Yes
$m \to b$	$c = 4/5$	No	$b, m \to c$	$c = 3/4$	No
	$\vdots$			$\vdots$	

Example

We have a bunch of baskets

$$B_1 = \{m, c, b\} \quad B_2 = \{m, p, j\} \quad B_3 = \{m, b\} \quad B_4 = \{c, j\}$$
$$B_5 = \{m, p, b\} \quad B_6 = \{m, c, b, j\} \quad B_7 = \{c, b, j\} \quad B_8 = \{b, c\}$$

- We have a minimum support $s = 3$ with confidence $c = 0.75$

Frequent itemsets

$\{b, m\}\{b, c\}$ $\{c, m\}$ $\{c, j\}$ $\{m, c, b\}$

Generate rules by eliminating anything below $c = 0.75$

Rule	Confidence	Remove	Rule	Confidence	Remove
$b \to m$	$c = 4/6$	Yes	$b, c \to m$	$c = 3/5$	Yes
$m \to b$	$c = 4/5$	No	$b, m \to c$	$c = 3/4$	No
$\vdots$			$\vdots$		

Other Similar Ideas about Frequent Itemsets

Maximal Frequent itemsets

No immediate superset is frequent

Closed itemsets

No immediate superset has the same count (> 0).

- It stores not only frequent information, but exact counts

Other Similar Ideas about Frequent Itemsets

Maximal Frequent itemsets

No immediate superset is frequent

Closed itemsets

No immediate superset has the same count (> 0).

- It stores not only frequent information, but exact counts

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Maximal

- $\{B\}$ is frequent but not maximal because superset $\{B, C\}$ also frequent.
- $\{A, B\}$ is frequent and maximal because its only superset $\{A, B, C\}$ is not.

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Maximal

- $\{B\}$ is frequent but not maximal because superset $\{B, C\}$ also frequent.
- $\{A, B\}$ is frequent and maximal because its only superset $\{A, B, C\}$ is not.

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Closed

- $\{C\}$ is frequent, but not closed because superset $\{B, C\}$ has same count.
- $\{B, C\}$ is frequent and closed because its only superset $\{A, B, C\}$ has smaller count.

Example: Maximal/Closed

Table

Set	Count	Maximal(S=3)	Closed
$\{A\}$	4	No	No
$\{B\}$	5	No	Yes
$\{C\}$	3	No	No
$\{A, B\}$	4	Yes	Yes
$\{A, C\}$	2	No	No
$\{B, C\}$	3	Yes	Yes
$\{A, B, C\}$	2	No	Yes

Closed

- $\{C\}$ is frequent, but not closed because superset $\{B, C\}$ has same count.
- $\{B, C\}$ is frequent and closed because its only superset $\{A, B, C\}$ has smaller count.

Outline

Computation Model

Now

- Back to finding frequent itemsets

Computation Model

Now

- Back to finding frequent itemsets

Computing Itemsets

- Typically, data is kept in flat files rather than in a database system.

Computation Model

Now

- Back to finding frequent itemsets

Computing Itemsets

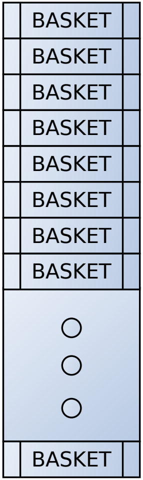
- Typically, data is kept in flat files rather than in a database system.

Thus

The File for the baskets

- It is stored on disk
- It is stored basket-by-basket

- You use k nested loops to generate all sets of size k



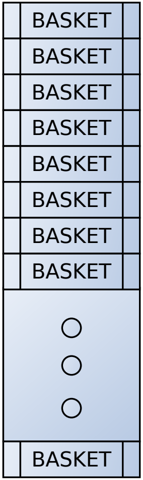
Thus

The File for the baskets

- It is stored on disk
- It is stored basket-by-basket

Baskets are small, but we have many baskets and many items

- You need to expand baskets into pairs, triples, etc. as you read the baskets
- You use k nested loops to generate all sets of size k



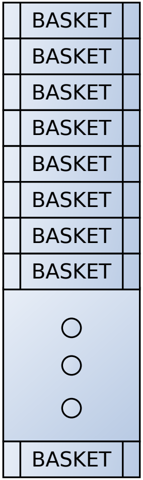
Thus

The File for the baskets

- It is stored on disk
- It is stored basket-by-basket

Baskets are small, but we have many baskets and many items

- You need to expand baskets into pairs, triples, etc. as you read the baskets
- You use k nested loops to generate all sets of size k



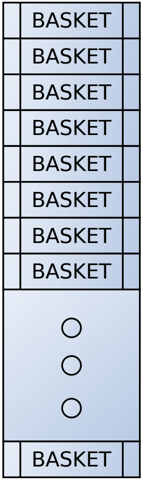
Thus

The File for the baskets

- It is stored on disk
- It is stored basket-by-basket

Baskets are small, but we have many baskets and many items

- You need to expand baskets into pairs, triples, etc. as you read the baskets
- You use k nested loops to generate all sets of size k



Then

We want to find frequent itemsets

- We want to find frequent itemsets.
- To find them, we have to count them.
- To count them, we have to generate them.

Then

We want to find frequent itemsets

- We want to find frequent itemsets.
- To find them, we have to count them.
- To count them, we have to generate them.

Then

We want to find frequent itemsets

- We want to find frequent itemsets.
- To find them, we have to count them.
- To count them, we have to generate them.

Computation Model

First

The true cost of mining disk-resident data is usually the number of disk I/O's.

Second

In practice, association-rule algorithms read the data in passes - all baskets are read in turn

Third

We measure the cost by the **number of passes** an algorithm makes over the data

Computation Model

First

The true cost of mining disk-resident data is usually the number of disk I/O's.

Second

In practice, association-rule algorithms read the data in passes - all baskets are read in turn

Third

We measure the cost by the number of passes an algorithm makes over the data

Computation Model

First

The true cost of mining disk-resident data is usually the number of disk I/O's.

Second

In practice, association-rule algorithms read the data in passes - all baskets are read in turn

Third

We measure the cost by the **number of passes** an algorithm makes over the data

Main-Memory Bottleneck I

The Main Problem

For many frequent-itemset algorithms, **main memory** is the critical resource.

- Because the combinatorial problem of calculating and counting the power set!!!

Main-Memory Bottleneck I

The Main Problem

For many frequent-itemset algorithms, **main memory** is the critical resource.

- Because the combinatorial problem of calculating and counting the power set!!!

As we read baskets

We need to count something, for example, occurrences of pairs of items.

Main-Memory Bottleneck I

The Main Problem

For many frequent-itemset algorithms, **main memory** is the critical resource.

- Because the combinatorial problem of calculating and counting the power set!!!

As we read baskets

We need to count something, for example, occurrences of pairs of items.

Main-Memory Bottleneck I

The Main Problem

For many frequent-itemset algorithms, **main memory** is the critical resource.

- Because the combinatorial problem of calculating and counting the power set!!!

As we read baskets

We need to count something, for example, occurrences of pairs of items.

Main-Memory Bottleneck II

Constraint

The number of different things we can count is limited by main memory.

Main-Memory Bottleneck II

Constraint

The number of different things we can count is limited by main memory.

Therefore

Swapping counts in/out is a disaster (why?).

Main-Memory Bottleneck II

Constraint

The number of different things we can count is limited by main memory.

Therefore

Swapping counts in/out is a disaster (why?).

Finding Frequent Pairs

Notice the following

The hardest problem often turns out to be finding the frequent pairs of items $\{i_1, i_2\}$

Finding Frequent Pairs

Notice the following

The hardest problem often turns out to be finding the frequent pairs of items $\{i_1, i_2\}$

Why?

Often frequent pairs are common, frequent triples are rare!!!

Finding Frequent Pairs

Notice the following

The hardest problem often turns out to be finding the frequent pairs of items $\{i_1, i_2\}$

Why?

Often frequent pairs are common, frequent triples are rare!!!

Probability of being frequent drops exponentially with size

Number of sets grows more slowly with size.

Finding Frequent Pairs

Notice the following

The hardest problem often turns out to be finding the frequent pairs of items $\{i_1, i_2\}$

Why?

Often frequent pairs are common, frequent triples are rare!!!

Probability of being frequent drops exponentially with size

Number of sets grows more slowly with size.

Thus

Let us first concentrate on pairs, then extend to larger sets.

Finding Frequent Pairs

The approach

- We always need to generate all the itemsets.
- But we would only like to count/keep track of those itemsets that in the end turn out to be frequent.

Finding Frequent Pairs

The approach

- We always need to generate all the itemsets.
- But we would only like to count/keep track of those itemsets that in the end turn out to be frequent.

Naïve Algorithm

What not to do

Naïve approach to finding frequent pairs

Naïve Algorithm

What not to do

Naïve approach to finding frequent pairs

What not to do

- Read file once, counting in main memory the occurrences of each pair:
  - From each basket of n items, generate its $\frac{n(n-1)}{2}$ pairs by two nested loops .

Naïve Algorithm

What not to do

Naïve approach to finding frequent pairs

What not to do

- Read file once, counting in main memory the occurrences of each pair:
  - From each basket of n items, generate its $\frac{n(n-1)}{2}$ pairs by two nested loops .

Naïve Algorithm

Fails if $(\text{Number of Items})^2$ exceeds main memory

Remember that the Number of Items can be $100\ Kb$ (Wal-Mart) or $10\ Gb$ (Web pages).

For example

Suppose we have 10^7 items and counts are 4-byte integers

Number of pairs of items

$$\frac{10^7\,(10^5\text{-}1)}{2} \approx 5 \times 10^{11} \tag{11}$$

Therefore, we need the following amount

$$4 \text{ bytes} \times 5 \times 10^{11} = 2 \times 10^{12} \text{ bytes} = 2 \text{ terabytes}$$

Naïve Algorithm

Fails if $(\text{Number of Items})^2$ exceeds main memory

Remember that the Number of Items can be $100\ Kb$ (Wal-Mart) or $10\ Gb$ (Web pages).

For example

Suppose we have 10^7 items and counts are 4-byte integers

Number of pairs of items

$$\frac{10^7\ (10^5\text{-}1)}{2} \approx 5 \times 10^{11} \quad (11)$$

Therefore, we need the following amount

$$4 \text{ bytes} \times 5 \times 10^{11} = 2 \times 10^{12} \text{ bytes} = 2 \text{ terabytes}$$

Naïve Algorithm

Fails if $(\text{Number of Items})^2$ exceeds main memory

Remember that the Number of Items can be $100\ Kb$ (Wal-Mart) or $10\ Gb$ (Web pages).

For example

Suppose we have 10^7 items and counts are 4-byte integers

Number of pairs of items

$$\frac{10^7\,(10^5\text{-}1)}{2} \approx 5 \times 10^{11} \tag{11}$$

Therefore, we need the following amount

$$4 \text{ bytes} \times 5 \times 10^{11} = 2 \times 10^{12} \text{ bytes} = 2 \text{ terabytes}$$

Naïve Algorithm

Fails if $(\text{Number of Items})^2$ exceeds main memory

Remember that the Number of Items can be $100\ Kb$ (Wal-Mart) or $10\ Gb$ (Web pages).

For example

Suppose we have 10^7 items and counts are 4-byte integers

Number of pairs of items

$$\frac{10^7\,(10^5\text{-}1)}{2} \approx 5 \times 10^{11} \tag{11}$$

Therefore, we need the following amount

$$4 \text{ bytes} \times 5 \times 10^{11} = 2 \times 10^{12} \text{ bytes} = 2 \text{ terabytes}$$

Counting Pairs in Memory

Approach 1 - Using a Triangular Matrix

- You can count all the pairs by simply using the counter at the cell $A[i, j] = A[i, j] + 1$.
- The storage used at this approach is 4 bytes per pair

- Plus some additional overhead for the hash table

Counting Pairs in Memory

Approach 1 - Using a Triangular Matrix

- You can count all the pairs by simply using the counter at the cell $A[i,j] = A[i,j] + 1$.
- The storage used at this approach is 4 bytes per pair

Approach 2 - Using an sparse array representation

Use a hash table of triples $[i,j,c] =$ "the count of the pair of items $\{i,j\}$ is c" using as index $i \circ j$.

- Plus some additional overhead for the hash table

Counting Pairs in Memory

Approach 1 - Using a Triangular Matrix

- You can count all the pairs by simply using the counter at the cell $A[i,j] = A[i,j] + 1$.
- The storage used at this approach is 4 bytes per pair

Approach 2 - Using an sparse array representation

Use a hash table of triples $[i,j,c] =$ "the count of the pair of items $\{i,j\}$ is c" using as index $i \circ j$.

Approach 2 - Using an sparse array representation

- If integers and item ids are 4 bytes, we need approximately 12 bytes for pairs with $count > 0$
- Plus some additional overhead for the hash table.

Counting Pairs in Memory

Approach 1 - Using a Triangular Matrix

- You can count all the pairs by simply using the counter at the cell $A[i,j] = A[i,j] + 1$.
- The storage used at this approach is 4 bytes per pair

Approach 2 - Using an sparse array representation

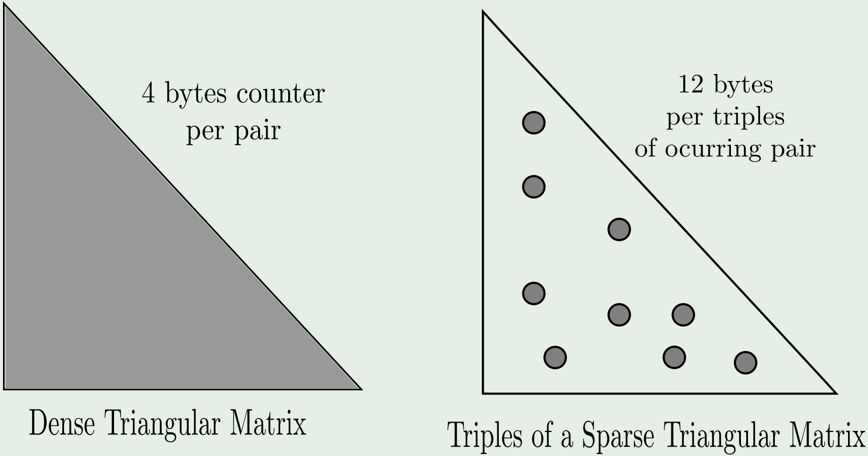
Use a hash table of triples $[i,j,c] =$ "the count of the pair of items $\{i,j\}$ is c" using as index $i \circ j$.

Approach 2 - Using an sparse array representation

- If integers and item ids are 4 bytes, we need approximately 12 bytes for pairs with $count > 0$
- Plus some additional overhead for the hash table.

Comparing the 2 Approaches

Dense vs Sparse



Triangular Matrix Approach

Triangular Matrix Approach

- n =total number items
- Count pair of items $\{i, j\}$ only if $i < j$

Triangular Matrix Approach

Triangular Matrix Approach

- n =total number items
- Count pair of items $\{i, j\}$ only if $i < j$

Storing Items in a Flat Array

- Keep pair counts in **lexicographic order:**
  - $\{1,2\}, \{1,3\}, ..., \{1,n\}, \{2,3\}, \{2,4\}, ..., \{2,n\}, \{3,4\}, ...$
- Pair $\{i, j\}$ is at position $(i{-}1)(n{-}i/2) + j{-}i$

Triangular Matrix Approach

Triangular Matrix Approach

- n =total number items
- Count pair of items $\{i, j\}$ only if $i < j$

Storing Items in a Flat Array

- Keep pair counts in **lexicographic order**:
  - $\{1,2\}, \{1,3\}, ..., \{1,n\}, \{2,3\}, \{2,4\}, ..., \{2,n\}, \{3,4\}, ...$
- Pair $\{i, j\}$ is at position $(i-1)(n-i/2) + j-i$

Triangular Matrix Approach

Triangular Matrix Approach

- n =total number items
- Count pair of items $\{i, j\}$ only if $i < j$

Storing Items in a Flat Array

- Keep pair counts in **lexicographic order**:
  - $\{1, 2\}, \{1, 3\}, ..., \{1, n\}, \{2, 3\}, \{2, 4\}, ..., \{2, n\}, \{3, 4\}, ...$
- Pair $\{i, j\}$ is at position $(i–1)(n–i/2) + j–i$

Triangular Matrix Approach

Triangular Matrix Approach

- n =total number items
- Count pair of items $\{i, j\}$ only if $i < j$

Storing Items in a Flat Array

- Keep pair counts in **lexicographic order**:
  - $\{1,2\},\{1,3\},...,\{1,n\},\{2,3\},\{2,4\},...,\{2,n\},\{3,4\},...$
- Pair $\{i, j\}$ is at position $(i–1)(n–i/2) + j–i$

Triangular Matrix Approach

Comparison

- Total number of pairs $n(n-1)/2$; total bytes$= 2n^2$
- Triangular Matrix requires 4 bytes per pair
- Approach 2 uses 12 bytes per pair (but only for pairs with count > 0)
  - It beats triangular matrix if less than 1/3 of possible pairs actually occur

Triangular Matrix Approach

Comparison

- Total number of pairs $n(n–1)/2$; total bytes$= 2n^2$
- Triangular Matrix requires 4 bytes per pair
- Approach 2 uses 12 bytes per pair (but only for pairs with count > 0)
  - It beats triangular matrix if less than $1/3$ of possible pairs actually occur

Triangular Matrix Approach

Comparison

- Total number of pairs $n(n-1)/2$; total bytes$=2n^2$
- Triangular Matrix requires 4 bytes per pair
- Approach 2 uses 12 bytes per pair (but only for pairs with count >0)
  - It beats triangular matrix if less than $1/3$ of possible pairs actually occur

Triangular Matrix Approach

Comparison

- Total number of pairs $n(n-1)/2$; total bytes$= 2n^2$
- Triangular Matrix requires 4 bytes per pair
- Approach 2 uses 12 bytes per pair (but only for pairs with count > 0)
  - **It beats triangular matrix if less than $1/3$ of possible pairs actually occur**

Observation About Using Triples

It is clear that

If we can store information in a hash table, we can really save memory.

However

False Positive Counts can increase because of the nature of the hash table.

IMPORTANT

Take this in consideration

Observation About Using Triples

It is clear that

If we can store information in a hash table, we can really save memory.

However

False Positive Counts can increase because of the nature of the hash table.

IMPORTANT

Take this in consideration

Observation About Using Triples

It is clear that

If we can store information in a hash table, we can really save memory.

However

False Positive Counts can increase because of the nature of the hash table.

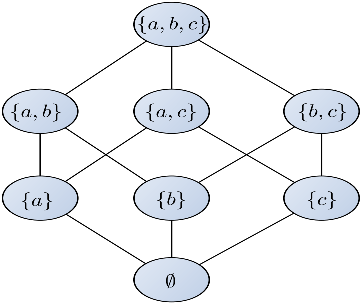
IMPORTANT

Take this in consideration

A-Priori Algorithm - (1)

The main algorithm idea

- A two-pass approach called a-priori limits the need for main memory



Key idea: monotonicity

- If a set of items I appears at least s times, so does every **subset** J of I.

Contrapositive for pairs

- If item i does not appear in s baskets, then no pair including i can appear in s baskets

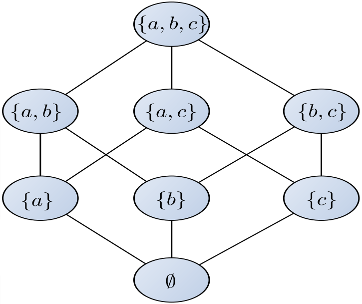
A-Priori Algorithm - (1)

The main algorithm idea

- A two-pass approach called a-priori limits the need for main memory

Key idea: monotonicity

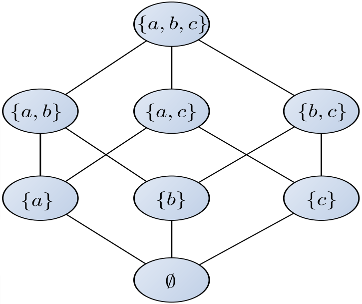
- If a set of items I appears at least s times, so does every **subset** J of I.



Contrapositive for pairs

- If item i does not appear in s baskets, then no pair including i can appear in s baskets

A-Priori Algorithm - (1)



The main algorithm idea

- A two-pass approach called a-priori limits the need for main memory

Key idea: monotonicity

- If a set of items I appears at least s times, so does every **subset** J of I.

Contrapositive for pairs

- If item i does not appear in s baskets, then no pair including i can appear in s baskets

Outline

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

Observation

- Items that appear at least s times are the frequent items

  - It requires memory proportional to square of frequent items only (for counts) i.e. $O(m^2)$
  - Plus a list of the frequent items (so you know what must be counted)

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

Observation

- Items that appear at least s times are the frequent items

Pass 2

- It read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - It requires memory proportional to square of frequent items only (for counts) i.e $O(n^2)$.
  - Plus a list of the frequent items (so you know what must be counted) .

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

Observation

- Items that appear at least s times are the frequent items

Pass 2

- It read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - It requires memory proportional to square of frequent items only (for counts) i.e $O(n^2)$.
  - Plus a list of the frequent items (so you know what must be counted) .

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

Observation

- Items that appear at least s times are the frequent items

Pass 2

- It read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - It requires memory proportional to square of frequent items only (for counts) i.e $O(n^2)$.
  - Plus a list of the frequent items (so you know what must be counted).

A-Priori Algorithm - (2)

Pass 1

- It reads baskets and count in main memory the occurrences of each individual item
  - It requires only memory proportional to $\#items$

Observation

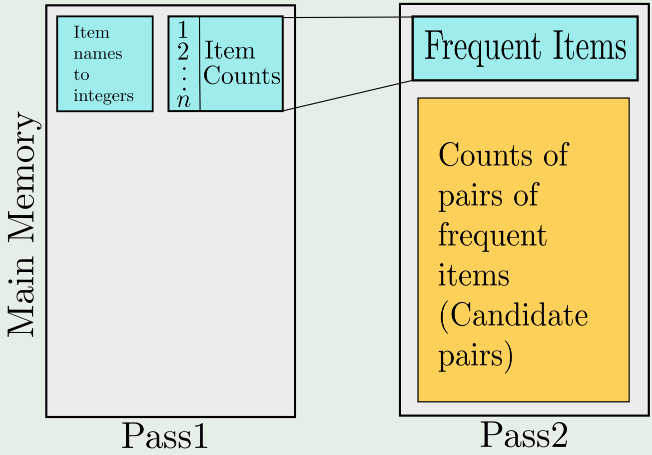
- Items that appear at least s times are the frequent items

Pass 2

- It read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - It requires memory proportional to square of frequent items only (for counts) i.e $O(n^2)$.
  - Plus a list of the frequent items (so you know what must be counted) .

Main-Memory Usage of the A-Priori Algorithm

Memory during the passes



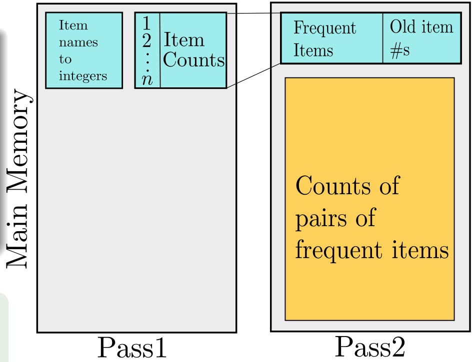
Details for A-Priori

What to do!!!

- You can use the triangular matrix method with $n =$ number of frequent items
  - It may save space compared with storing triples

After That

- Create a new numbering for the frequent items by generating an array (frequent items table) with entries $1, 2, ..., n$
- In addition an extra table that relates the new numbers with the original item numbers.



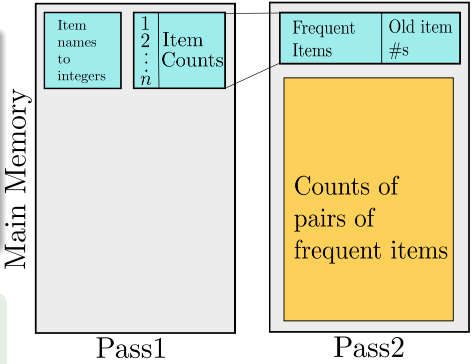
Details for A-Priori

What to do!!!

- You can use the triangular matrix method with $n =$ number of frequent items
  - It may save space compared with storing triples

After That

- Create a new numbering for the frequent items by generating an array (frequent items table) with entries $1, 2, ..., n$
- In addition an extra table that relates the new numbers with the original item numbers.



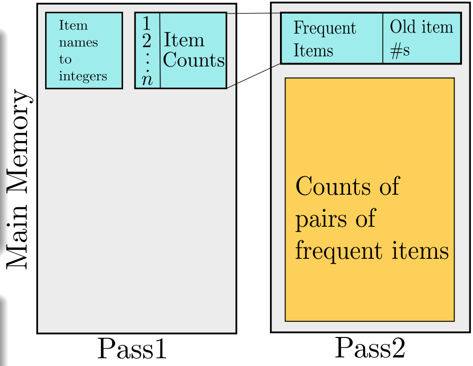
Details for A-Priori

What to do!!!

- You can use the triangular matrix method with $n =$ number of frequent items
  - It may save space compared with storing triples

After That

- Create a new numbering for the frequent items by generating an array (frequent items table) with entries $1, 2, ..., n$
- In addition an extra table that relates the new numbers with the original item numbers.



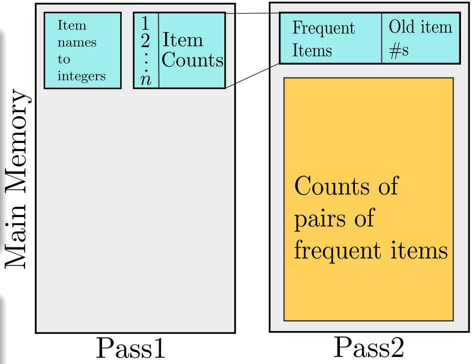
Details for A-Priori

What to do!!!

- You can use the triangular matrix method with $n =$ number of frequent items
  - It may save space compared with storing triples

After That

- Create a new numbering for the frequent items by generating an array (frequent items table) with entries $1, 2, ..., n$
- In addition an extra table that relates the new numbers with the original item numbers.



Mechanic for The Second Step

First

For each basket, look in the frequent-items table to see which of its items are frequent.

Second

In a double loop, generate all pairs of frequent items in that basket.

Third

For each such pair, add $+1$ to its count in the data structure used to store counts.

Mechanic for The Second Step

First

For each basket, look in the frequent-items table to see which of its items are frequent.

Second

In a double loop, generate all pairs of frequent items in that basket.

Third

For each such pair, add $+1$ to its count in the data structure used to store counts.

Mechanic for The Second Step

First

For each basket, look in the frequent-items table to see which of its items are frequent.

Second

In a double loop, generate all pairs of frequent items in that basket.

Third

For each such pair, add $+1$ to its count in the data structure used to store counts.

Outline

Frequent Triples, Etc.

We have then the following procedure for k-tuples

- For each k, we construct two sets of k-tuples (sets of size k):
  - C_k = candidate k-**tuples** = those that might be frequent sets (support $> s$) based on information from the pass for k–1
  - L_k = the set of truly frequent k-tuples

Frequent Triples, Etc.

We have then the following procedure for k-tuples

- For each k, we construct two sets of k-tuples (sets of size k):
  - C_k = candidate k-**tuples** = those that might be frequent sets (support $> s$) based on information from the pass for k–1
  - L_k = the set of truly frequent k-tuples

Flow Diagram

Frequent Triples, Etc.

We have then the following procedure for k-tuples

- For each k, we construct two sets of k-tuples (sets of size k):
  - C_k = candidate k-**tuples** = those that might be frequent sets (support $> s$) based on information from the pass for k–1
  - L_k = the set of truly frequent k-tuples

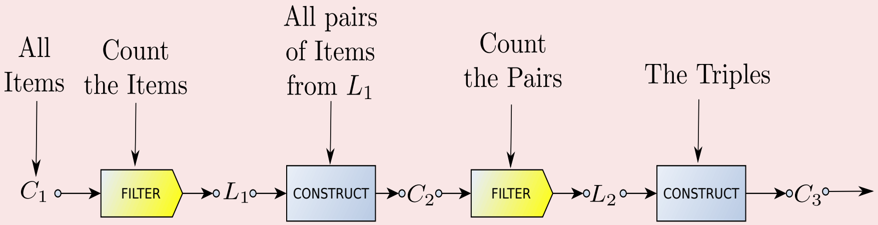
Flow Diagram

Frequent Triples, Etc.

We have then the following procedure for k-tuples

- For each k, we construct two sets of k-tuples (sets of size k):
  - C_k = candidate k-**tuples** = those that might be frequent sets (support $> s$) based on information from the pass for k–1
  - L_k = the set of truly frequent k-tuples

Flow Diagram



Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{ \ \{b\} \ \{c\} \ \{j\} \ \{m\} \ \{n\} \ \{p\} \ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\} \ \{b, j\} \ \{b, m\} \ \{c, j\} \ \{c, m\} \ \{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\} \ \{b, c\} \ \{c, m\} \ \{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\} \ \{b, c, j\} \ \{b, m, j\} \ \{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\ \{b, j\}\ \{b, m\}\ \{c, j\}\ \{c, m\}\ \{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\ \{b, c\}\ \{c, m\}\ \{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\ \{b, c, j\}\ \{b, m, j\}\ \{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\ \{b, j\}\ \{b, m\}\ \{c, j\}\ \{c, m\}\ \{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\ \{b, c\}\ \{c, m\}\ \{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\ \{b, c, j\}\ \{b, m, j\}\ \{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\, \{b, j\}\, \{b, m\}\, \{c, j\}\, \{c, m\}\, \{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\, \{b, c\}\, \{c, m\}\, \{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\, \{b, c, j\}\, \{b, m, j\}\, \{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\,\{b, j\}\,\{b, m\}\,\{c, j\}\,\{c, m\}\,\{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\,\{b, c\}\,\{c, m\}\,\{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\,\{b, c, j\}\,\{b, m, j\}\,\{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b,c\}\,\{b,j\}\,\{b,m\}\,\{c,j\}\,\{c,m\}\,\{j,m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b,m\}\,\{b,c\}\,\{c,m\}\,\{c,j\}\}$
- Generate $C_3 = \{\{b,c,m\}\,\{b,c,j\}\,\{b,m,j\}\,\{c,m,j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b,c,m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\ \{b, j\}\ \{b, m\}\ \{c, j\}\ \{c, m\}\ \{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\ \{b, c\}\ \{c, m\}\ \{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\ \{b, c, j\}\ \{b, m, j\}\ \{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\,\{b, j\}\,\{b, m\}\,\{c, j\}\,\{c, m\}\,\{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\,\{b, c\}\,\{c, m\}\,\{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\,\{b, c, j\}\,\{b, m, j\}\,\{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

Example

Hypothetical steps of the A-Priori algorithm

- C1 $= \{\ \{b\}\ \{c\}\ \{j\}\ \{m\}\ \{n\}\ \{p\}\ \}$
- Count the support of itemsets in C_1
- Prune non-frequent: $L_1 = \{b, c, j, m\}$
- Generate $C_2 = \{\{b, c\}\,\{b, j\}\,\{b, m\}\,\{c, j\}\,\{c, m\}\,\{j, m\}\}$
- Count the support of itemsets in C_2
- Prune non-frequent: $L_2 = \{\{b, m\}\,\{b, c\}\,\{c, m\}\,\{c, j\}\}$
- Generate $C_3 = \{\{b, c, m\}\,\{b, c, j\}\,\{b, m, j\}\,\{c, m, j\}\}$
- Count the support of itemsets in C_3
- Prune non-frequent: $L_3 = \{\{b, c, m\}\}$

A-Priori for All Frequent Itemsets

Properties

- One pass for each k (itemset size)
- Needs room in main memory to count each candidate k–tuple
- For typical market-basket data and reasonable support (e.g., 1%), $k = 2$ requires the most memory

A-Priori for All Frequent Itemsets

Properties

- One pass for each k (itemset size)
- Needs room in main memory to count each candidate k–tuple
- For typical market-basket data and reasonable support (e.g., 1%), $k = 2$ requires the most memory

A-Priori for All Frequent Itemsets

Properties

- One pass for each k (itemset size)
- Needs room in main memory to count each candidate k–tuple
- For typical market-basket data and reasonable support (e.g., 1%), $k = 2$ requires the most memory

Still Problems with Memory

This happens

When counting the candidates in C_2.

Still Problems with Memory

This happens

When counting the candidates in C_2.

Can we reduce the use of Memory?

Is this even possible?

Note: Actually in PCY, this is removed altogether!!!

Still Problems with Memory

This happens

When counting the candidates in C_2.

Can we reduce the use of Memory?

Is this even possible?

Yes, if we are willing to live under uncertain terms!!!

Remember the collisions at the hash tables!!!

Note: Actually in PCY, this is removed altogether!!!

Still Problems with Memory

This happens

When counting the candidates in C_2.

Can we reduce the use of Memory?

Is this even possible?

Yes, if we are willing to live under uncertain terms!!!

Remember the collisions at the hash tables!!!

Note Actually in PCY, this is removed altogether!!!

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory

- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory

- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory

- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory

- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!

PCY (Park-Chen-Yu) Algorithm

Observation

In pass 1 of a-priori, most memory is idle

- We store only individual item counts
- Can we use the idle memory to reduce memory required in pass 2?

Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory

- Keep a count for each bucket into which pairs of items are hashed
  - Just the count, not the pairs that hash to the bucket!

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2.   for each item $i_i \in t_k$
3.   add 1 to item's count
4.   for each pair of items:
5.   Hash the pair into a bucket in the hash table
6.   Add 1 to the counter at that bucket

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2.   for each item $i_i \in t_k$
3.   add 1 to item's count
4.   for each pair of items:
5.   Hash the pair into a bucket in the hash table
6.   Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2. for each item $i_i \in t_k$
3. add 1 to item's count
4. for each pair of items:
5. Hash the pair into a bucket in the hash table
6. Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2.   for each item $i_i \in t_k$
3.   add 1 to item's count
4.   for each pair of items:
5.   Hash the pair into a bucket in the hash table
6.   Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2.   for each item $i_i \in t_k$
3.   add 1 to item's count
4.   for each pair of items:
5.   Hash the pair into a bucket in the hash table
6.   Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2. for each item $i_i \in t_k$
3. add 1 to item's count
4. for each pair of items:
5. Hash the pair into a bucket in the hash table
6. Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

PCY Algorithm - First Pass

Algorithm

1. for each basket t_k:
2. for each item $i_i \in t_k$
3. add 1 to item's count
4. for each pair of items:
5. Hash the pair into a bucket in the hash table
6. Add 1 to the counter at that bucket

Note

Pairs of items need to be generated from the input file because they are not present in the file

At Pass 1, we introduce uncertainty

By using the hash table

Yes, COLLISIONS!!!

That means that it is possible that pairs $\{i, j\}$ and $\{t, l\}$

They can hash to the same bucket.

At Pass 1, we introduce uncertainty

By using the hash table

Yes, COLLISIONS!!!

That means that it is possible that pairs $\{i, j\}$ and $\{t, l\}$

They can hash to the same bucket.

We want the following

What?

We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.

We want the following

What?

We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.

We generate candidate pairs $\{i, j\}$ such that

1. i and j are frequent items.
2. $\{i, j\}$ hashes to a frequent bucket.

We want the following

What?

We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.

We generate candidate pairs $\{i, j\}$ such that

1. i and j are frequent items.
2. $\{i, j\}$ hashes to a frequent bucket.

We want the following

What?

We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.

We generate candidate pairs $\{i, j\}$ such that

1. i and j are frequent items.
2. $\{i, j\}$ hashes to a frequent bucket.

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation

- But, for a bucket with total count less than s, none of its element pairs can be frequent
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of two frequent items)

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation

- But, for a bucket with total count less than s, none of its element pairs can be frequent
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of two frequent items)

Pass 2

- Only count pairs that hash to frequent buckets

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation

- But, for a bucket with total count less than s, none of its element pairs can be frequent
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of two frequent items)

Pass 2

- Only count pairs that hash to frequent buckets

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation

- But, for a bucket with total count less than s, none of its element pairs can be frequent
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of two frequent items)

Pass 2

- Only count pairs that hash to frequent buckets

Observation about Buckets

Something Notable

- If a bucket contains a frequent pair, then the bucket is surely frequent
  - But we cannot use the hash to eliminate any member of this bucket
- Even without any frequent pair, a bucket can still be frequent

Observation

- But, for a bucket with total count less than s, none of its element pairs can be frequent
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of two frequent items)

Pass 2

- Only count pairs that hash to frequent buckets

PCY Algorithm - Between Passes

Replace the buckets by a bit-vector (Bloom Filter Style)

- 1 means the bucket count exceeded the support s (a frequent bucket) and 0 means it did not

Property 1

4-byte integer counts are replaced by bits, so the bit-vector requires $1/32$ of memory

Property 2

Also, decide which items are frequent and list them for the second pass

PCY Algorithm - Between Passes

Replace the buckets by a bit-vector (Bloom Filter Style)

- 1 means the bucket count exceeded the support s (a frequent bucket) and 0 means it did not

Property 1

4-byte integer counts are replaced by bits, so the bit-vector requires $1/32$ of memory

Property 2

Also, decide which items are frequent and list them for the second pass

PCY Algorithm - Between Passes

Replace the buckets by a bit-vector (Bloom Filter Style)

- 1 means the bucket count exceeded the support s (a frequent bucket) and 0 means it did not

Property 1

4-byte integer counts are replaced by bits, so the bit-vector requires $1/32$ of memory

Property 2

Also, decide which items are frequent and list them for the second pass

PCY Algorithm - Pass 2

First

- Count all pairs $\{i, j\}$ that meet the conditions for being a candidate pair:
  1. Both i and j are frequent items
  2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1 (i.e., frequent bucket)

PCY Algorithm - Pass 2

First

- Count all pairs $\{i, j\}$ that meet the conditions for being a candidate pair:
  1. Both i and j are frequent items
  2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1 (i.e., frequent bucket)

Thus

- Both conditions are necessary for the pair to have a chance of being frequent

PCY Algorithm - Pass 2

First

- Count all pairs $\{i, j\}$ that meet the conditions for being a candidate pair:
  1. Both i and j are frequent items
  2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1 (i.e., frequent bucket)

Thus

- Both conditions are necessary for the pair to have a chance of being frequent

PCY Algorithm - Pass 2

First

- Count all pairs $\{i, j\}$ that meet the conditions for being a candidate pair:
  1. Both i and j are frequent items
  2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1 (i.e., frequent bucket)

Thus

- Both conditions are necessary for the pair to have a chance of being frequent

We Minimize the Uncertainty

How

By the two checkings!!!

We Minimize the Uncertainty

How

By the two checkings!!!

First one

1. Both i and j are frequent items

Here certain amount of uncertainty is accepted in order to reduce the amount of memory used.

We Minimize the Uncertainty

How

By the two checkings!!!

First one

1. Both i and j are frequent items

Second one

The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1.

Here certain amount of uncertainty is accepted in order to reduce the amount of memory used

We Minimize the Uncertainty

How

By the two checkings!!!

First one

1. Both i and j are frequent items

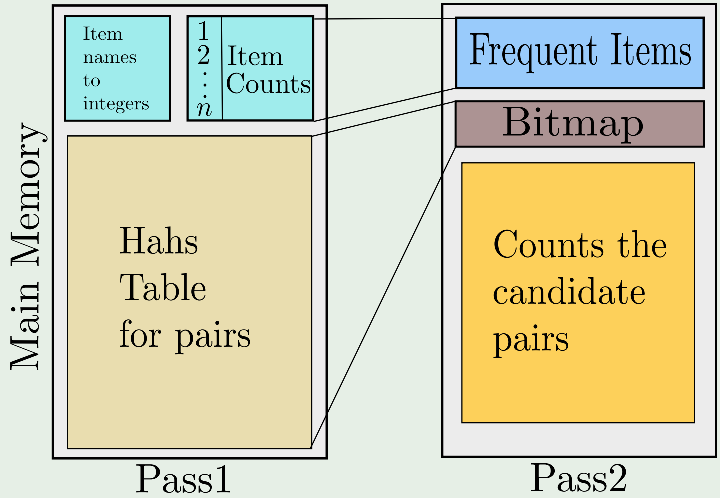
Second one

The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1.

Here certain amount of uncertainty is accepted in order to reduce the amount of memory used

Main-Memory: Picture of PCY

Something Notable



Main-Memory Details

Buckets require a few bytes each

- Note: we do not have to count past s
- Number of *buckets* is O(main-memory size)

Main-Memory Details

Buckets require a few bytes each

- Note: we do not have to count past s
- Number of $buckets$ is O(main-memory size)

Table of triples

- On second pass, a table of $(item, item, count)$ triples is essential (we cannot use triangular matrix approach, why?)
  - Thus, hash table must eliminate approx. $2/3$ of the candidate pairs for PCY to beat a-priori.

Main-Memory Details

Buckets require a few bytes each

- Note: we do not have to count past s
- Number of $buckets$ is O(main-memory size)

Table of triples

- On second pass, a table of $(item, item, count)$ triples is essential (we cannot use triangular matrix approach, why?)
  - Thus, hash table must eliminate approx. $2/3$ of the candidate pairs for PCY to beat a-priori.

Main-Memory Details

Buckets require a few bytes each

- Note: we do not have to count past s
- Number of $buckets$ is O(main-memory size)

Table of triples

- On second pass, a table of $(item, item, count)$ triples is essential (we cannot use triangular matrix approach, why?)
  - Thus, hash table must eliminate approx. $2/3$ of the candidate pairs for PCY to beat a-priori.

Outline

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Refinement: Multistage Algorithm

Limit the number of candidates to be counted

- Remember: Memory is the bottleneck
- Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent

Key idea

After Pass 1 of PCY, rehash only those pairs that **qualify** for Pass 2 of PCY

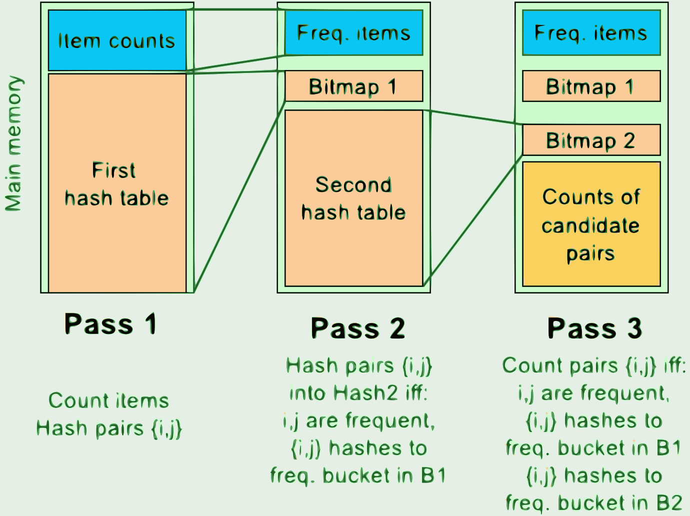
- i and j are frequent
- $\{i, j\}$ hashes to a frequent bucket in the first hash table

Then

- On middle pass, fewer pairs contribute to buckets, so fewer false positives
  - By hashing $\{i, j\}$ to a frequent bucket in the second hash table.
- Requires 3 passes over the data

Main-Memory: Multistage

Something Notable



Multistage - Pass 3

Thus

- Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.

Multistage - Pass 3

Thus

- Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.

Multistage - Pass 3

Thus

- Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.

Multistage - Pass 3

Thus

- Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.

Important Points

First

The two hash functions have to be independent

Important Points

First

The two hash functions have to be independent

Second

We need to check both hashes on the third pass

- If not, we would end up counting pairs of frequent items that hashed first to an infrequent bucket but happened to hash second to a frequent bucket

Important Points

First

The two hash functions have to be independent

Second

We need to check both hashes on the third pass

- If not, we would end up counting pairs of frequent items that hashed first to an infrequent bucket but happened to hash second to a frequent bucket

Important Points

We can reduce collision

They have independent hash functions.

Thus, the probability of false positive is reduced because independence

$$P\left(\text{Collision by hash 1,Collision by hash 2}\right) = P\left(\text{Collision by hash 1}\right) \times \ldots$$
$$P\left(\text{Collision by hash 2}\right)$$

Important Points

We can reduce collision

They have independent hash functions.

Thus, the probability of false positive is reduced because independence

$$P\left(\text{ Collision by hash 1,Collision by hash 2 }\right) = P\left(\text{Collision by hash 1}\right) \times ... \\ P\left(\text{Collision by hash 2}\right)$$

Outline

Refinement: Mulitihash

Key idea

- Use several independent hash tables on the first pass

Refinement: Mulitihash

Key idea

- Use several independent hash tables on the first pass

Risk

- Halving the number of buckets doubles the average count
  - We have to be sure most buckets will still not reach count s

Refinement: Mulitihash

Key idea

- Use several independent hash tables on the first pass

Risk

- Halving the number of buckets doubles the average count
  - We have to be sure most buckets will still not reach count s

- If so, we can get a benefit like multistage, but in only 2 passes

Refinement: Mulitihash

Key idea

- Use several independent hash tables on the first pass

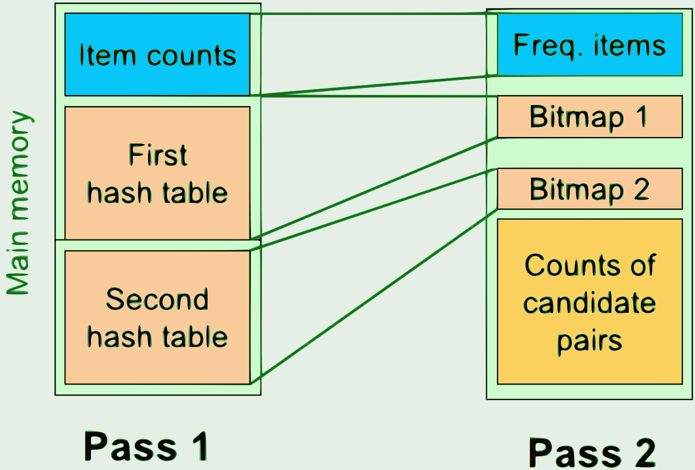
Risk

- Halving the number of buckets doubles the average count
  - We have to be sure most buckets will still not reach count s

- If so, we can get a benefit like multistage, but in only 2 passes

Main-Memory: Mulitihash

Something Notable



PCY: Extensions

Multistage or Multihash

- Either multistage or multihash can use more than two hash functions

Multistage

- In multistage, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory

Multihash

- For multihash, the bit-vectors occupy exactly what one PCY bitmap does, but too many hash functions makes all counts $> s$

PCY: Extensions

Multistage or Multihash

- Either multistage or multihash can use more than two hash functions

Multistage

- In multistage, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory

Multihash

- For multihash, the bit-vectors occupy exactly what one PCY bitmap does, but too many hash functions makes all counts $> s$

PCY: Extensions

Multistage or Multihash

- Either multistage or multihash can use more than two hash functions

Multistage

- In multistage, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory

Multihash

- For multihash, the bit-vectors occupy exactly what one PCY bitmap does, but too many hash functions makes all counts$> s$

Frequent Itemsets in ≤ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Frequent Itemsets in $\leq$ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Question

- Can we use fewer passes?

  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Frequent Itemsets in ≤ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Question

- Can we use fewer passes?

Use 2 or fewer passes for all itemset sizes

- Use 2 or fewer passes for all itemset sizes, but may miss some frequent itemsets
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Frequent Itemsets in ≤ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Question

- Can we use fewer passes?

Use 2 or fewer passes for all itemset sizes

- Use 2 or fewer passes for all itemset sizes, but may miss some frequent itemsets
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Frequent Itemsets in ≤ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Question

- Can we use fewer passes?

Use 2 or fewer passes for all itemset sizes

- Use 2 or fewer passes for all itemset sizes, but may miss some frequent itemsets
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Frequent Itemsets in ≤ 2 Passes

k Passes

- A-Priori, PCY, etc., take k passes to find frequent itemsets of size k.

Question

- Can we use fewer passes?

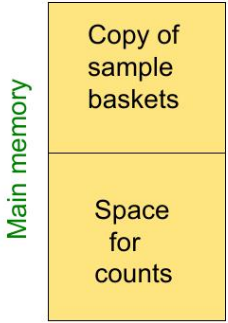
Use 2 or fewer passes for all itemset sizes

- Use 2 or fewer passes for all itemset sizes, but may miss some frequent itemsets
  - Random sampling
  - SON (Savasere, Omiecinski, and Navathe)
  - Toivonen

Random Sampling (1)

Thus

- Take a random sample of the market baskets

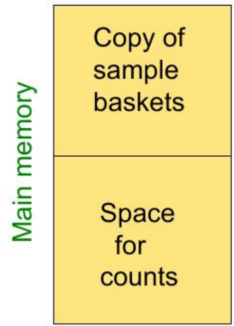


Random Sampling (1)

Thus

- Take a random sample of the market baskets

- Run A-priori or one of its improvements in main memory
  - So we do not pay for disk I/O each time we increase the size of itemsets
  - Reduce support threshold proportionally to match the sample size

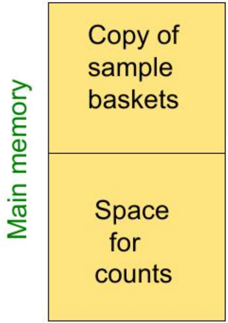


Random Sampling (1)

Thus

- Take a random sample of the market baskets

- Run A-priori or one of its improvements in main memory
  - So we do not pay for disk I/O each time we increase the size of itemsets
  - Reduce support threshold proportionally to match the sample size

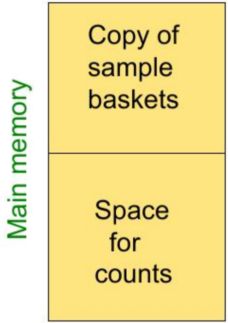


Random Sampling (1)

Thus

- Take a random sample of the market baskets

- Run A-priori or one of its improvements in main memory
  - So we do not pay for disk I/O each time we increase the size of itemsets
  - Reduce support threshold proportionally to match the sample size



However

Problem

We still have the problem of the false positives!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

However

- You do not catch sets frequent in the whole but in the sample
  - Solution
    - Smaller threshold, e.g., ps (p fraction size sample), helps catch more truly frequent itemsets.
    - Again you can do the solution 1, but you need more memory!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

However

- You do not catch sets frequent in the whole but in the sample
  - Solution
    - Smaller threshold, e.g., ps (p fraction size sample), helps catch more truly frequent itemsets.
    - Again you can do the solution 1, but you need more memory!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

However

- You do not catch sets frequent in the whole but in the sample
  - Solution
    - Smaller threshold, e.g., ps (p fraction size sample), helps catch more truly frequent itemsets.
    - Again you can do the solution 1, but you need more memory!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

However

- You do not catch sets frequent in the whole but in the sample
  - Solution
    - ★ Smaller threshold, e.g., ps (p fraction size sample), helps catch more truly frequent itemsets.
    - ★ Again you can do the solution 1, but you need more memory!!!

Solution

Solution 1

- Verify that the candidate pairs are truly frequent in the entire data set by a second pass (This avoids false positives)
- You need more memory!!!

However

- You do not catch sets frequent in the whole but in the sample
  - Solution
    - Smaller threshold, e.g., ps (p fraction size sample), helps catch more truly frequent itemsets.
    - Again you can do the solution 1, but you need more memory!!!

SON (Savasere, Omiecinski, Navathe) Algorithm - (1)

Repeatedly read small subsets

- Repeatedly read small subsets of the baskets into main memory and run an in-memory algorithm to find all frequent itemsets
  - Note: we are not sampling, but processing the entire file in memory-sized chunks

SON (Savasere, Omiecinski, Navathe) Algorithm - (1)

Repeatedly read small subsets

- Repeatedly read small subsets of the baskets into main memory and run an in-memory algorithm to find all frequent itemsets
  - Note: we are not sampling, but processing the entire file in memory-sized chunks

Itemset becomes a candidate

- An itemset becomes a candidate if it is found to be frequent in any one or more subsets of the baskets.

SON (Savasere, Omiecinski, Navathe) Algorithm - (1)

Repeatedly read small subsets

- Repeatedly read small subsets of the baskets into main memory and run an in-memory algorithm to find all frequent itemsets
  - Note: we are not sampling, but processing the entire file in memory-sized chunks

Itemset becomes a candidate

- An itemset becomes a candidate if it is found to be frequent in any one or more subsets of the baskets.

SON Algorithm - (2)

Second pass

- On a second pass, count all the candidate itemsets and determine which are frequent in the entire set .

Key "monotonicity" idea

- an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.

SON Algorithm - (2)

Second pass

- On a second pass, count all the candidate itemsets and determine which are frequent in the entire set .

Key “monotonicity” idea

- an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.

SON Distributed Version

- SON lends itself to distributed data mining

SON Distributed Version

- SON lends itself to distributed data mining

Baskets distributed among many nodes

- Compute frequent itemsets at each node
- Distribute candidates to all nodes
- Accumulate the counts of all candidates

SON Distributed Version

- SON lends itself to distributed data mining

Baskets distributed among many nodes

- Compute frequent itemsets at each node
- Distribute candidates to all nodes
- Accumulate the counts of all candidates

SON Distributed Version

- SON lends itself to distributed data mining

Baskets distributed among many nodes

- Compute frequent itemsets at each node
- Distribute candidates to all nodes
- Accumulate the counts of all candidates