Data Stream Mining- Lecture 6

Frequent Pattern Mining in Data Streams

Chandresh Kumar Maurya, Research Assistant Professor

Eötvös Loránd University, Budapest, Hungary

November 12, 2019

What is frequent pattern mining?

Definition

Frequent pattern mining is the identification of most frequent pattern in a collection of data.

An example:

Transactions	Itemsets
t_1	{milk, bread, butter}
t_2	{milk, beer, diper}
t_3	{diper, shampoo coke}
t_4	{beer, diper, milk, bread}
t_5	{beer, diper, coke}
t_6	{milk, diper, shampoo,beer}

Some preliminary

- A collection of items is denoted by $A = \{a_1, a_2, \ldots, a_m\}$. E.g. items could be things you buy from supermarket, pages you visit over internet in one session, etc.
- Any subset I of items A i.e. $I \subseteq A$ is called an **itemset**.
- A set of transactions is denoted by $T = \{t_1, t_2, \ldots, t_n\}$ called **transaction database**.
- The **support** of an itemset defined as the number of transactions containing it and denoted by σ.

Some Definitions

A frequent itemset mining can be formally defined as:

Definition

Given a set of items $A = \{a_1, a_2, \ldots, a_m\}$ and a vector of transactions $T = \{t_1, t_2, \ldots, t_n\}$ and *minimum* support σ_{min} the frequent itemset mining problem is: To find the set of frequent itemsets, i.e., the set $\{I \subseteq A | \sigma(I) \geq \sigma_{min}\}$

Itemset mining example

TID	*Item set*
1	$\{a, d, e\}$
2	$\{b, c, d\}$
3	$\{a, c, e\}$
4	$\{a, c, d, e\}$
5	$\{a, e\}$
6	$\{a, c, d\}$
7	$\{b, c\}$
8	$\{a, c, d, e\}$
9	$\{b, c, e\}$
10	$\{a, d, e\}$

0 items	1 item	2 items	3 items
$\emptyset$: 10	$\{a\}$: 7	$\{a, c\}$: 4	$\{a, c, d\}$: 3
	$\{b\}$: 3	$\{a, d\}$: 5	$\{a, c, e\}$: 3
	$\{c\}$: 7	$\{a, e\}$: 6	$\{a, d, e\}$: 4
	$\{d\}$: 6	$\{b, c\}$: 3	
	$\{e\}$: 7	$\{c, d\}$: 4	
		$\{c, e\}$: 4	
		$\{d, e\}$: 4	

Figure: Enumeration of all possible frequent itemsets with $\sigma_{min} = 30\%$

Search Space

- How many itemsets are possible over the set A?
- Anti Monotone property: If $X, Y \subseteq I$ are two itemsets such that $X \subseteq Y \implies \sigma(Y) \leq \sigma(X)$.
- What above means is that if an itemset is infrequent, then all of its superset will also be infrequent.
- *Apriori* [Agrawal et al., 1994] was the first algorithm to study frequent itemset mining problem.

Algorithms for FP in the streaming case

Frequent itemset mining in data streams is challenging due to several reasons:

1. Low memory
2. Single pass
3. Huge size of search space
4. Previously infrequent item become frequent and vice versa.

Three approahces:

- Approaches which don't distinguish old and new itemsets (landmark window)
- Approaches which give more weights to recent items
- Itemset mining over multiple time granularity

Lossy Counting Algorithm [Manku and Motwani, 2002a]

- Is an algorithm for frequency counting of items (single item stream) over data streams
- Needs two user-specified parameters. 1) support threshold $s \in [0, 1]$ and 2) error threshold $\epsilon \in [0, 1]$ such that $\epsilon \leq s$.

The algorithm gives the following guarantees:

1. All items with true frequencies greater than sN are output and no *false negative*
2. All items with true frequency less than $(s - \epsilon)N$ are not output.
3. estimate frequency are less than true frequency by at most ϵN.

Where N is the data stream size.

Contd...

Suppose you are interested in finding frequency of items with support $s = 1\%$. Further let your error margin be $\epsilon = 0.1\%$. Then

- From Property (1), all items with frequency of 1% will be output and no *false negative*.
- From property (2), all items with frequency less than 0.9% will not output. This leaves items with frequency in $[0.9, 1)$. They may or may not be output. Those which are output are false positive.
- From (3), all individual frequencies are less than true frequencies by at most 0.1%.

The Algorithm- definitions

- Incoming streams is divided into buckets of width $w = \lceil 1/\epsilon \rceil$.
- Buckets are labeled with bucket *ids* starting from 1.
- Denote current bucket id by $b_{current}$ and $= \lceil N/w \rceil$.
- True frequency of item e is f_e.
- We store frequent items in a data structure D which stores entries of the form (e, f, Δ). Where
  - e is the element
  - f is current (running) frequency of e.
  - Δ is the maximum possible error in f

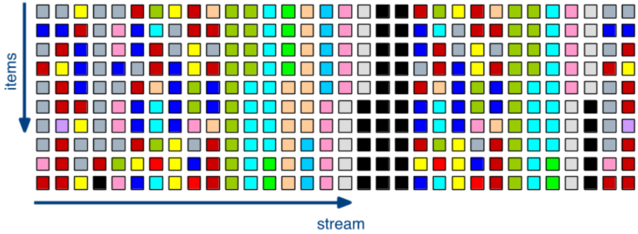
Contd..

Intuition

- Initially D is empty.
- As an item arrives, we consult the database D. If there is an entry, we increment its frequency f by 1.
- Otherwise, create a new entry in D by inserting a tuple of $(e, 1, b_{current} - 1)$.
- From time to time, elements from D are also deleted as follows: An entry from D is deleted such that $f + \Delta \leq b_{current}$.
- When a user requets for a list of items with s, we output those entries in D satisfying $f \geq (s - \epsilon)N$.
- Number of counters needed in the worst case $\frac{1}{\epsilon}\log(\epsilon N)$

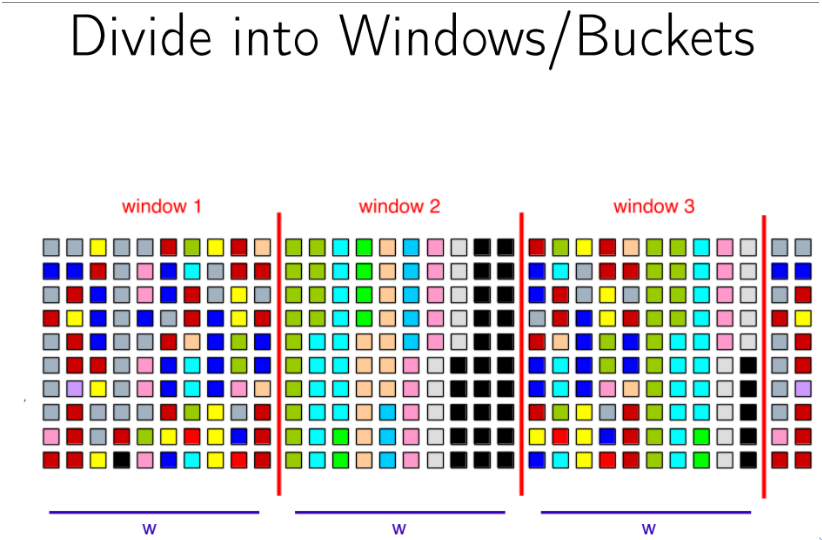
Illustration (slides adapted from [Manku and Motwani, 2002b])

Lossy Counting in Action



Incoming Stream of Colours

Illustration



Illustration

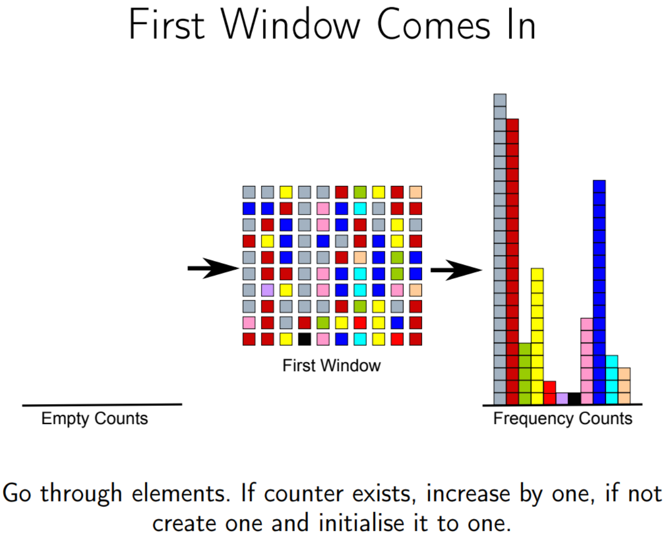
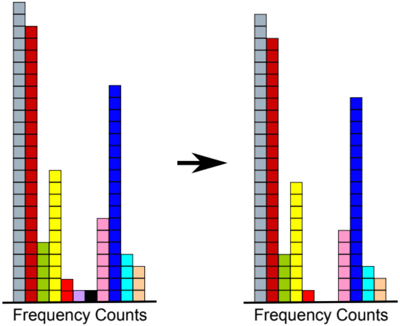


Figure: Lossy counting in action

Illustration

Adjust Counts at Window Boundaries



Reduce all counts by one. If counter is zero for a specific element, drop it.

Figure: Lossy counting in action

Illustration

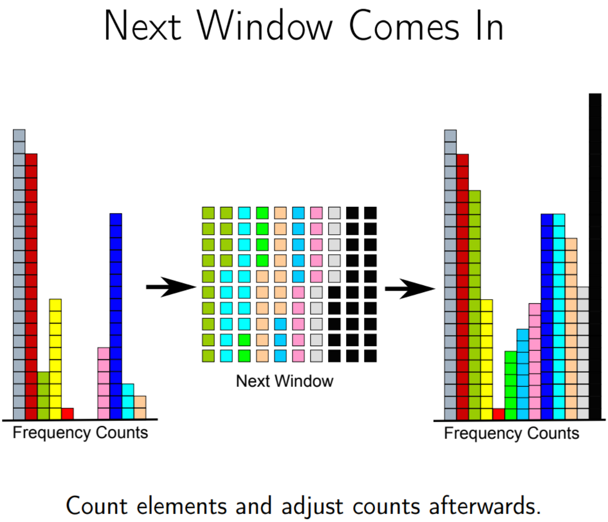
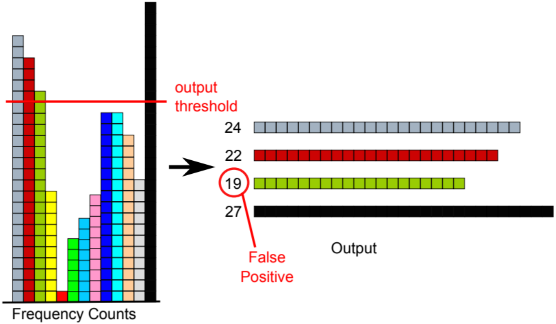


Figure: Lossy counting in action

Illustration

Output

With $s = 10\%, \epsilon = 1\%, N = 200$



To reduce false positives to acceptable amount, only output counters with frequency $f > (s - \epsilon)N = 18$.

Figure: Lossy counting in action

Sticky Sampling

Main ideas:

- Sticky sampling algorithm is probabilistic version of the lossy algorithm.
- Besides support s, error tolerance ϵ for frequency count, it requires probability of failure δ as user parameter.
- Worst-case space need for counter is independent of the stream size N, so is applicable for infinite stream.

The Algorithm

- Initially data structure D is empty.
- For each incoming element/item, if an entry exists, add (e, f) to D after incrementing its frequency by 1.
- If the item is misisng in D, then you sample it at a rate r which is 1 initially. If the item is selected by sampling, we add $(e, 1)$ to D. Otherwise ignore it.
- Smapling rate r varies over time as follows: first t elements are sampled at a rate $r = 1$, next $2t$ elements are sampled at $r = 2$, next $4t$ elements are sampled at $r = 4$, and so on.
- Whenever the sampling rate changes, we also scan entries in D , and update them as follows: For each entry (e, f), we repeatedly toss an unbiased coin until the coin toss is successful, diminishing f by one for every unsuccessful outcome; if f becomes 0 during this process, we delete the entry from D.
- Worst-case space complexity $\frac{1}{\epsilon} \log(\frac{1}{s\delta})$.

Frequent itemsets mining

How to modify lossy algorithm for counting frequency of itemsets?
Challenges:

- Single-pass
- Low memory
- Varibale sized trasnactions
- Avoid enumerating all possible subsets of items.

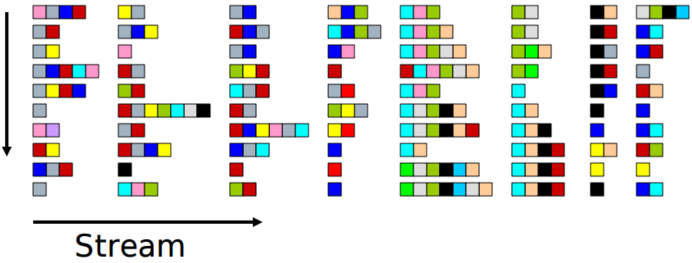
Contd...

Key ideas:

- Denote database by D, initially empty. D contains entries of the form (set, f, Δ), where *set* is subsets of items, and (f, Δ) carry their old meaning.
- As before, inputs stream is divided into buckets labeled with *bucket ids* starting from 1. Current bucket is $b_{current}$.
- Instead of processing each transaction as it arrives, we store as many transactions as possible into main memory and process *batch-wise*. Let β denote the batch size.
- D is updated as follows:
  - $UPDATE\_SET$: For each entry $(set, f, \Delta) \in D$, update D by couting frequency of the set in the current batch.If the udpated entry satisfies $f + \Delta \leq b_{current}$, delete this.
  - NEW_SET: If a set *set* has frequency $f \geq \beta$ in the current batch, and it is not in D, create a new entry $(set, f, b_{current} - \beta)$

Illustration

Frequent Itemsets Problem ...



➤ Identify all subsets of items whose current frequency exceeds s = 0.1%.

Frequent Itemsets => Association Rules

Figure: Lossy counting for frequent sets of items

Sequence pattern mining

What is a sequence?

Sequence pattern mining

What is a sequence?

Definition

A sequence is an *ordered* list of items. Formally, a sequence of items $A = \{a_1, a_2, a_3, \ldots, a_n\}$ is such that items in the set are ordered according to some rule. Rule could be timestamp based on arrival, or values of the items.

Sequence pattern mining

What is a sequence?

Definition

A sequence is an *ordered* list of items. Formally, a sequence of items $A = \{a_1, a_2, a_3, \ldots, a_n\}$ is such that items in the set are ordered according to some rule. Rule could be timestamp based on arrival, or values of the items.

An example:

A *transaction database*

TID	itemsets
10	a, b, d
20	a, c, d
30	a, d, e
40	b, e, f

A *sequence database*

SID	sequences
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Applications

Sequence pattern mining is found in daily life. E.g.

1. Shopping of items: first buy computer, then pen-drive, then keyboard over a period of time.
2. Some event sequence: earthquake$\rightarrow$ sunami$\rightarrow$medical treatment
3. DNA/protein sequences etc.

What Is Sequential Pattern Mining?

- Given a set of sequences and support threshold, find the complete set of *frequent* subsequences

A *sequence* : < (ef) (ab) (df) c b >

A *sequence database*

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

<a(bc)dc> is a *subsequence* of <a(abc)(ac)d(cf)>

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

Challenges on Sequential Pattern Mining

- A huge number of possible sequential patterns are hidden in databases
- A mining algorithm should
  - find the complete set of patterns, when possible, satisfying the minimum support (frequency) threshold
  - be highly efficient, scalable, involving only a small number of database scans
  - be able to incorporate various kinds of user-specific constraints

Studies on Sequential Pattern Mining

- Concept introduction and an initial Apriori-like algorithm
  - Agrawal & Srikant. Mining sequential patterns, [ICDE'95]
- Apriori-based method: GSP (Generalized Sequential Patterns: Srikant & Agrawal [EDBT'96])
- Pattern-growth methods: FreeSpan & PrefixSpan (Han et al.KDD'00; Pei, et al. [ICDE'01])
- Vertical format-based mining: SPADE (Zaki [Machine Leanining'00])
- Constraint-based sequential pattern mining (SPIRIT: Garofalakis, Rastogi, Shim [VLDB'99]; Pei, Han, Wang [CIKM'02])
- Mining closed sequential patterns: CloSpan (Yan, Han & Afshar [SDM'03])

Bibliography I

Agrawal, R., Srikant, R., et al. (1994).
Fast algorithms for mining association rules.
In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

Manku, G. and Motwani, R. (2002a).
Approximate frequency counts over streaming data.
In *Proc. 28th Int. Conf. Very Large Data Bases*, pages 346–357.

Manku, G. and Motwani, R. (2002b).
Frequency counts over data streams.