Data Stream Mining

Péter Kiss, Teaching Assistant

Eötvös Loránd University, Budapest, Hungary

October 7, 2020

Data Stream

Definition

A data stream is an ordered (not necessarily always) and potentially infinite sequence of data points.

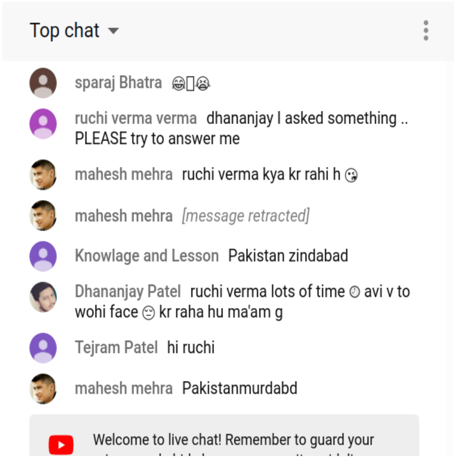
$$x_1, x_2, x_3, \ldots,$$

where x_is are tuples constituted of numbers, words, sequences, etc.

Such streams are ubiquitous and we are always around them. For example:

- Click streams
- Sensor measurements
- Satellite imaging data
- Power grid electricity distribution
- Banking/e-commerce transactions

Examples of Data Streams :



Top chat

sparaj Bhatra 😂▯😭

ruchi verma verma dhananjay I asked something .. PLEASE try to answer me

mahesh mehra ruchi verma kya kr rahi h 😘

mahesh mehra *[message retracted]*

Knowlage and Lesson Pakistan zindabad

Dhananjay Patel ruchi verma lots of time 🕐 avi v to wohi face ☹ kr raha hu ma'am g

Tejram Patel hi ruchi

mahesh mehra Pakistanmurdabd

Welcome to live chat! Remember to guard your

Figure: Youtube live comments

Data Stream Mining Algorithms

Under stream mining algorithms we usually involve some summarization of the stream into a model.
Tasks related to data streams (planned):

- **Sampling and Filtering**
- Counting distinct elements
- estimating moments
- Clustering
- Frequent pattern mining
- Change/Concept drift detection
- Time series

Literature

Rajaraman, Anand, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2011.
Gama, Joao. Knowledge discovery from data streams. CRC Press, 2010.

Characteristics of data streams

A data stream has several distinguishing features such as:

- Unbounded Size
    - Transient (that means it lasts for only few seconds or minutes)
    - Single-pass over data
    - Only summaries can be stored
    - Real-time processing (in-memory)
- Data streams are not static
    - Incremental algorithms: incorporate new data into the model
    - Some models : decremental updates - discard influence of data points if they are outdated/outliers
    - Concept Drifts - Temporal order may be important

Adaptive algoritms

Desirable properties for learning high speed, time-changing data streams:

- Incrementality
- on-line learning
- constant time to process
- single scan over training set
- Ability to handle concept drift
- limited computational resources
- anytime protocol -
- support for distributed data gathering and processing

Querying streams

Standing queries

Permanently executed, the process itself have been designed to answer them.
Examples - Ocean-surface temperature:

1. Alert when temperature exceeds 25 Centigrade
2. Alert when average of most recent 24 measurements exceeds 25 Centigrade
3. Maximum temperature ever recorded

Querying streams

Ad-hoc queries

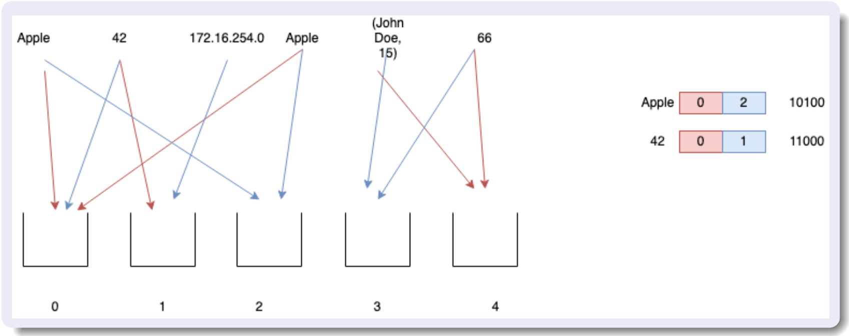
A question asked once about the current state of a stream. If we do not store all the data items we have seen, we cannot expect to be able to answer arbitrary queries.

Techniques to answer

If we have some idea however what kind of queries may be asked we can prepare for them by storing appropriate summaries of streams.

1. Sliding windows - collect all the data arrived in a given time period/or a range lof most recent data
2. Sampling - maintain an appropriate representation of all data.

Hashing


Apple
42
172.16.254.0
Apple
(John Doe, 15)
66
0
1
2
3
4
Apple 0 2 10100
42 0 1 11000

Hashing

"Hashset" for example in Java

Definition

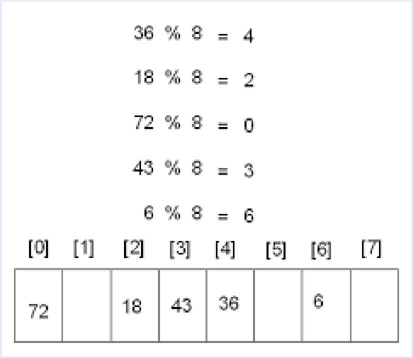
A hash function h takes some value (of any data type) the so-called *hash-key*, and it assigns it to one of B *buckets* (that is produces a *bucket number*).

The key requirement of a hash function is that if *hash-keys* are drawn randomly from a population, h should send similar number of values to each of the B buckets.

Hashing

Example : Hashing integer keys

simple way is the division method : $h(x) = k \mod B$, that returns the remainder when x divided by B.

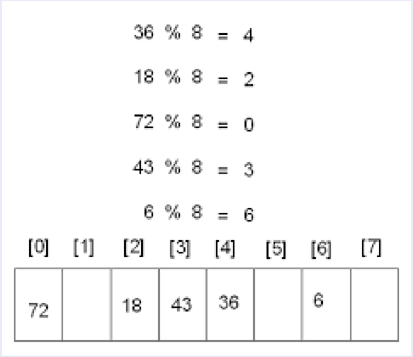


source and more methods: `http://faculty.cs.niu.edu/~freedman/340/340notes/340hash.htm`

Hashing

Example : Hashing integer keys

simple way is the division method : $h(x) = k \mod B$, that returns the remainder when x divided by B.



Problem

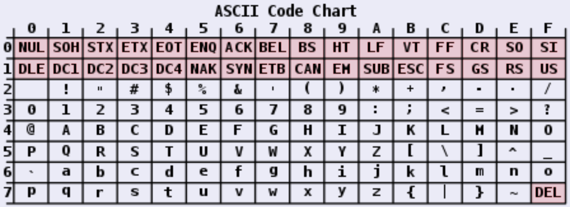
What if the 4th element is 44?

Hashing

Hashing non-integer keys

All data type have values that always can be translated to bits, and bit series always can be interpreted as integers.

ASCII



ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

String hashing

For small B: sum the values

For big B: concatenate digits

Motivation

General Problem

selecting a subset of a stream so that we can ask ad-hoc queries about the selected subset and have the answers be statistically representative of the stream as a whole.

Example:
Domain : Stream of queries arrived at a search engine
Query: What fraction of user's queries are repeated?

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question?

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question? Idea: save each search with 10% probability. Law of Large Numbers $\rightarrow$ about $1/10$ of the searches of each particular user will be stored.

Repeated search example

Let us assume that we can store 10% of the searches. How can we answer the question? Idea: save each search with 10% probability. Law of Large Numbers $\rightarrow$ about $1/10$ of the searches of each particular user will be stored.

s searches once , d twice, and no searches more times

The correct answer:

$$\frac{\text{no. of double search-queries}}{\text{all search queries}} = \frac{d}{(d+s)} \tag{1}$$

Repeated search example

But what we get with our method?

Repeated search example

But what we get with our method? Probabilities of a single search appears in summary $1/10$
Probability of double one appears once :
$1/10 \cdot 9/10 + 9/10 \cdot 1/10 = 18/100$
Probability that both occurrence of a double search appears:
$1/10 \cdot 1/10$

Thus the result:

$$\frac{d/100}{s/10 + 18d/100 + d/100} = \frac{d}{(10s + 19d)} \neq \frac{d}{d+s} \tag{2}$$

what does that mean?
The sample was NOT REPRESENTATIVE!!

Repeated search example

How we obtain a representative sample?
Instead of taking the proportion of searches, take all the searches $1/10$ part of the users!
When a new user appears generate a random number from 0 to 9, and if it let us say 0 then store the queries of the user.

Repeated search example

How we obtain a representative sample?
Instead of taking the proportion of searches, take all the searches $1/10$ part of the users!
When a new user appears generate a random number from 0 to 9, and if it let us say 0 then store the queries of the user.
What is the problem with this?

Repeated search example

What is the problem with this?
How many users can we store, and how much time does it takes to decide whether the given user's activity is recorded?

Repeated search example

What is the problem with this?
How many users can we store, and how much time does it takes to decide whether the given user's activity is recorded?
Solution: HASH the user names into 10 buckets, and if it goes to bucket 0, store the search query.

General Sampling Problem

Fixed fraction sample

A stream consists of tuples of some components (user query, time for example), and some subset of these components are *key* components.
Our goal is to get a representative sample of some size a/b. Then hash key value x for each tuple into b buckets, and store the tuple if $h(x) < a$.

Maintain sample of bounded size with time:

Very large number of buckets B, and threshold variable t. At the beginning $t = B - 1$. store tuple if $h(K) < t$. If allocated space is about to run out , throw away all tuples with $h(K) = t$, then $t - -$.

Filtering

Problem

Select interesting data points from a huge stream.

Filtering

Problem

Select interesting data points from a huge stream.

Easy

The easy problem when the selection criterion can be calculated, for example the tuple has an attribute with a given value.

Filtering

Problem

Select interesting data points from a huge stream.

Easy

The easy problem when the selection criterion can be calculated, for example the tuple has an attribute with a given value.

Hard

The hard problem: criterion based on membership in a group.

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?
Ans: **Bloom Filter**

Example : Checking if a user-id is present

Can you guess how gmail checks if a user-id is available?
Ans: **Bloom Filter**
The key idea: Maintain the following:

- A bit-array of length n.
- A number of hash functions $h_1(), h_2(), \ldots, h_k()$.
- A set S of m key values.

Intuation

We iterate over all the members at training, and hash each one of them onto the array. Any bit of the array, that has been produced by any hash function for any element will be set to 1. Thus if we test an element, and if it maps to all 1s we assume, that it is a member of S, otherwise we can be certain that it is not.

Bloom Filter

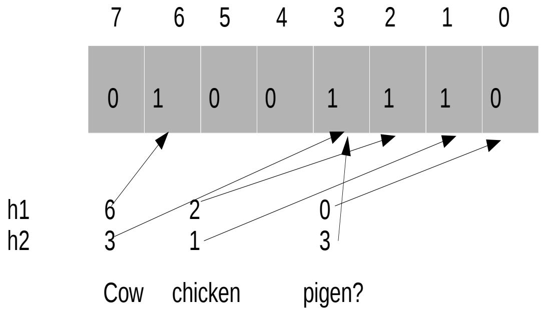


Figure: Hashing items in a bit-array. h_1 and h_2 are hash functions.

Bloom filter can answer if an item passes through it or not. That is, it gives **guaranteed answer** if a user id is available and probabilistic answer if a user-id is NOT available by measuring the collision. If at least one bit is not set, user-id is available and if all bits are set, it means either user-id *may* be available.

Limitations of bloom filter

1. Size of the bloom filter is very important. Smaller bloom filter, larger false positive (FP) and vice versa.
2. Number of hash functions? Larger number of hash functions, quicker the bloom filter fills as well as slow filter. Too few hash functions, too many FPs unless all are *good* hash functions.

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
After throwing all the darts, how many target we expect to be hit at least once?

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
After throwing all the darts, how many target we expect to be hit at least once?
Probabilities:

- given dart does not hit a given target:

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
After throwing all the darts, how many target we expect to be hit at least once?
Probabilities:

- given dart does not hit a given target: $\frac{(x-1)}{x}$ (the probability that one of the other target is hit)
- none of the y darts git a given target:

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
After throwing all the darts, how many target we expect to be hit at least once?
Probabilities:

- given dart does not hit a given target: $\frac{(x-1)}{x}$ (the probability that one of the other target is hit)
- none of the y darts git a given target: $(\frac{(x-1)}{x})^y$

$(\frac{(x-1)}{x})^y = (1-\frac{1}{x})^{x\frac{y}{x}}$. With $(1-\epsilon) \approx 1/e$ for small ϵ:
$(1-\frac{1}{x})^{x\frac{y}{x}} = e^{-y/x}$

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
Found that the probability of no darts hitting a particular target: $e^{-y/x}$

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function.
Intuition: y darts thrown at x targets, with any darts equally likely to hit any target.
Found that the probability of no darts hitting a particular target: $e^{-y/x}$
Each of the n bit of the filter is a target, and each k*hash value* of the m member of S is a dart. $(y = k \cdot m$, and $x = n)$ The probability of a bit is

- 0 is the same as it got no hit : $e^{-km/n}$
- 1 is the same as it got at least one hit : $1 - e^{-km/n}$

Limitations of bloom filter

Probability of false positives

Limitations of bloom filter

Probability of false positives in function of n bit array length, m the number of members of S, and k number of hash function. We want the probability for any bit is 0 to be as high as possible, otherwise the probability for nonmember element (email address) will be hashed do any 0 elemnt will be too low. That is, if it maps to all 0-s we will say that it is already occupied and reject

Contd...

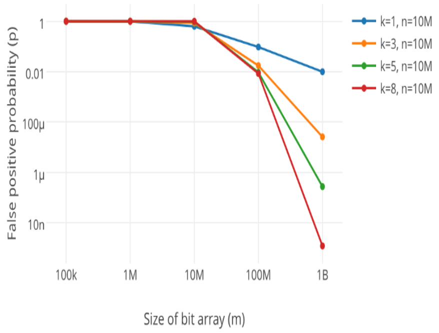


Figure: Number of hash functions vs FPR. Source: https://www.semantics3.com/blog/use-the-bloom-filter-luke-b59fd0839fc4/

Bibliography I