Web Data Mining

Lecture 11: Mining Data Streams

Jaroslav Kuchař & Milan Dojčinovski
jaroslav.kuchar@fit.cvut.cz, milan.dojchinovski@fit.cvut.cz

Czech Technical University in Prague - Faculty of Information Technologies - Software and Web Engineering

Summer semester 2019/2020
Humla v0.3

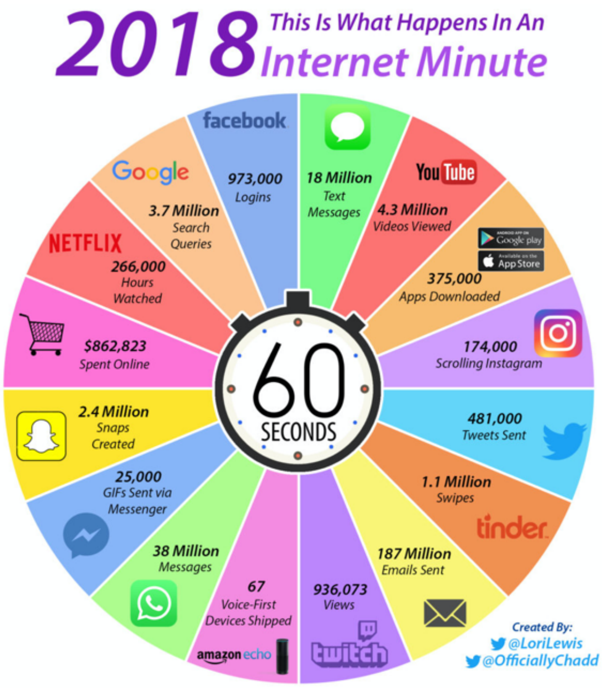
Overview

- Introduction
- Stream Data Processing
- Approaches
- Technologies

Introduction

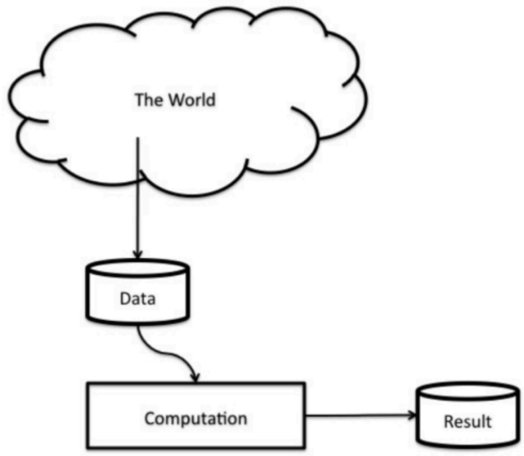
- Motivation
  - *Traditional approaches assume to have all data available in the database*
  - *Data is not static*
    - → *Data is growing all the time*
    - → *Every second new data is generated*
    - → *Data arrives in a stream and if it is not processed immediately or stored, then it is lost forever.*
- Application context
  - *When your data is too large to fit in the memory*
  - *When new data is constantly being generated, and/or is dependent upon time*
- Examples
  - *Web data*
  - *Sensor data*
  - *Telcos, activity data, Social networks, Real-time recommendations, Fraud or Spam detection, ...*

Internet Minute



Data Mining Overview

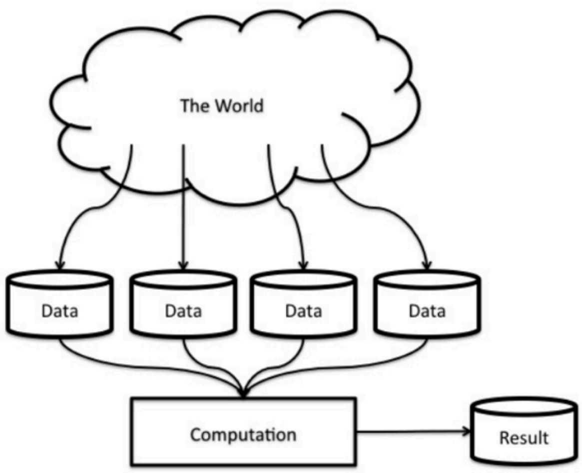
- Single Machine Data Mining



* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

Distributed Storage

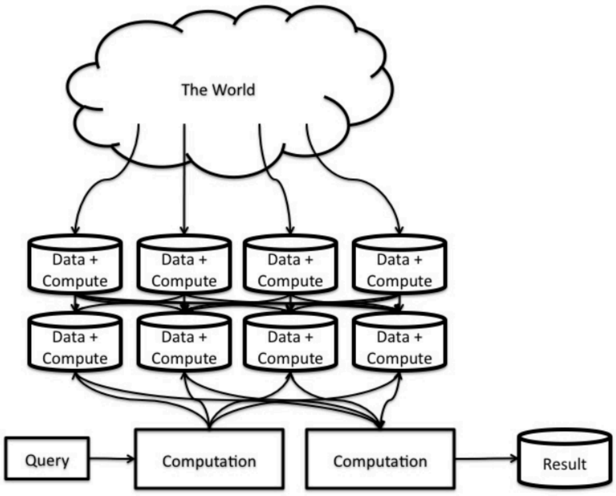
- Distributed Storage



* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

Distributed Computation

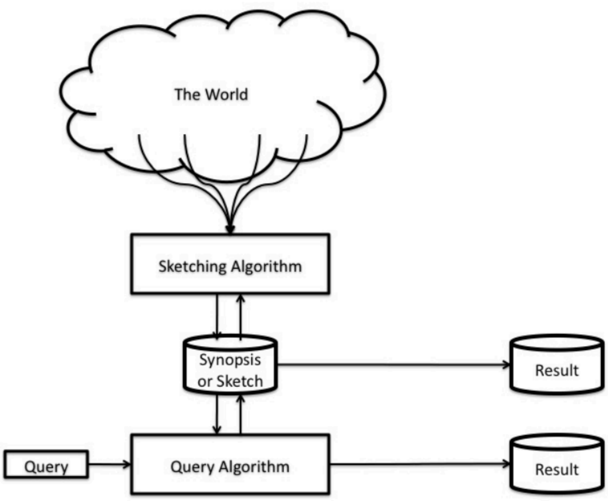
- Distributed Computation



* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

Streaming Model

- Streaming Model



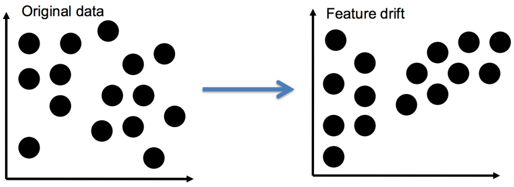
* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

Data Streams

- Characteristics
  - *High volume (possibly infinite) of continuous data*
  - *Data arrive at a rapid rate*
  - *Data distribution changes on the fly*
  - *Limited access to historical data*
  - *Can't store them all*
- Static vs Streaming data
  - *"If the data distribution is stable, mining a data stream is largely the same as mining a large data set, since statistically we can draw and mine a sufficient sample"*
- Challenges
  - *The data is evolving*
  - *Finding and understanding changes*
  - *Maintaining an updated model*

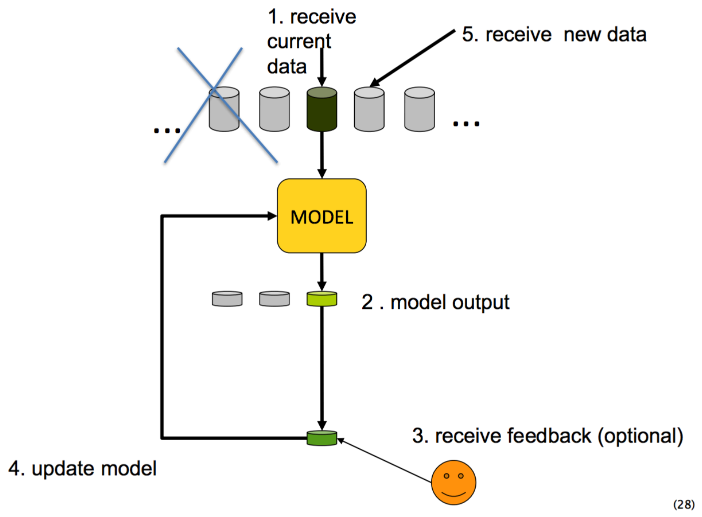
Types of changes in Data

- Evolving/drifting data = data distribution changes over time
  - *Feature drift*
    → distribution of input data X changes
  - *Real concept drift*
    → relation between input X and target y changes
  - *Arrival of new information*
- Examples
  - *Feature drift*



Mining Streaming Data

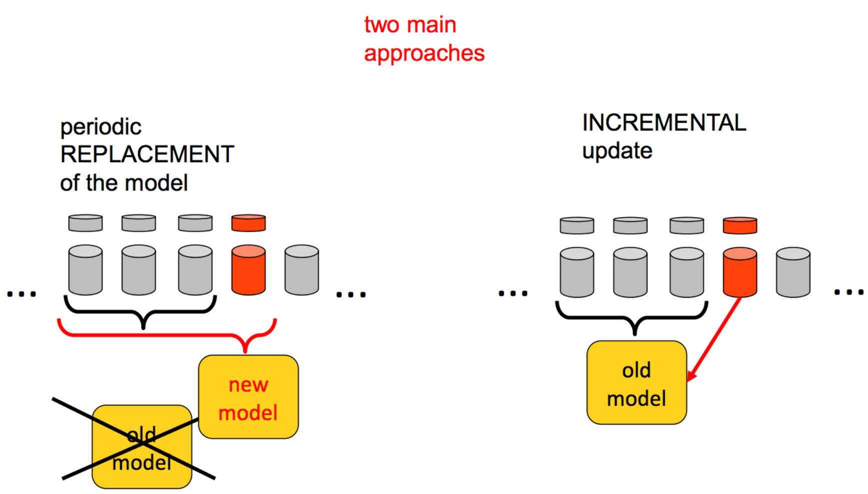
- Mining streaming data



* A. Bifet, J. Gama, R. Gavalda, G. Krempl, M. Pechenizkiy, B. Pfahringer, M. Spiliopoulou, I. Zliobaite: Advanced Topics on Data Stream Mining -

Model Adaptation

- Model Adaptation



* A. Bifet, J. Gama, R. Gavalda, G. Krempl, M. Pechenizkiy, B. Pfahringer, M. Spiliopoulou, I. Zliobaite: Advanced Topics on Data Stream Mining -

Computing in Data Streams

- Axioms
  - *One pass*
  - *Low time per item - read, process, discard*
  - *Sublinear memory - only summaries or sketches*
  - *Anytime, real-time answers*
  - *The stream evolves over time*
- Characteristics
  - *Approximate answers are often OK*
  - *Algorithms use a source of independent random bits*
    $\rightarrow$ *So different runs give different outputs*
  - *But "most runs" are "approximately correct"*

Overview

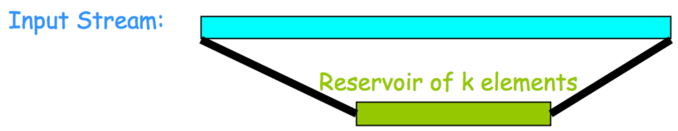
- Introduction
- Stream Data Processing
- Approaches
- Technologies

Sampling

- Sampling
  - *Find uniform random samples of an infinite data stream*
- Input
  - *Stream of data that arrive online*
  - *Sample size k*
  - *Sample range*
    - → *entire stream*
    - → *most recent window (count-based or time-based)*
- Output
  - *k elements chosen uniformly at random within the sample range*

Reservoir Sampling

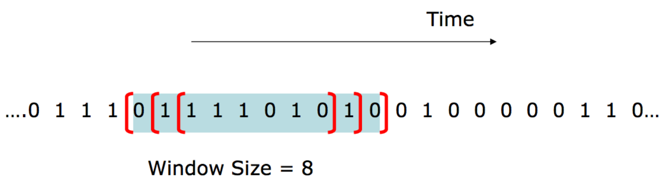
- Classical algorithm
  - *Goal: maintains a fixed-size uniform random sample*
  - *Approach*
    - → *Put the first k elements from the stream into the repository*
    - → *When the i-th element arrives*
      - → *Add it to reservoir S with probability p*
      - → *If added, randomly remove an element from S*
- Duplicates
  - *Stream contains duplicate elements*
  - *Any value occurring frequently in the sample is a wasteful use of the available space*



* Haixun Wang, Jian Pei, Philip S. Yu: Online Mining of Data Streams: Problems, Applications and Progress.

Moving Window

- Moving Window
  - *Timeliness*
    → old data are not useful
  - *Restrict samples to a window of recent data*
    → As new data arrives, old data "expires"
  - *Reservoir sampling cannot handle data expiration*
- Naive algorithm
  - *Place a moving window of size N on the stream*
  - *an old element y expires when a new element x arrives*



Other Processing

- Sketches
  - *Sampling techniques and sliding window models focus on a small part of the data*
  - *Sketches try to summarize the entire data, often at multiple levels of detail*
    → build a small-space summary for a distribution vector (e.g., histogram)
- Randomized Algorithms
  - *mainly consider Monte Carlo algorithms*
    → has bounds on the running time but may not return the correct result
  - *may err with some small, but controllable, probability.*

Overview

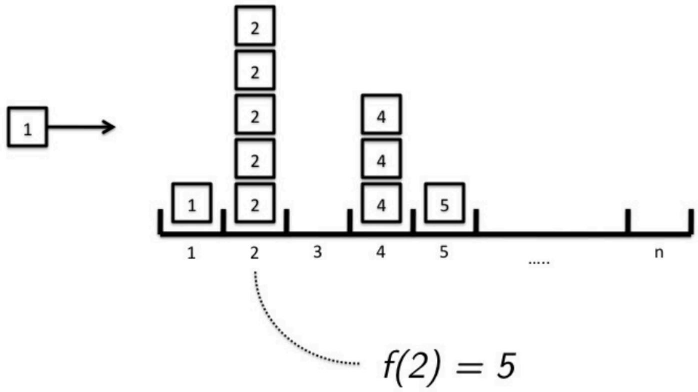
- Introduction
- Stream Data Processing
- Approaches
- Technologies

Approaches

- Tasks
  - *Trivial*
    $\longrightarrow$ *count items, sum values, find min/max, sample*
  - *Nontrivial*
    $\longrightarrow$ *alert on new item, most frequent item, finding median*
- Examples of
  - *Item frequencies*
  - *Count distinct items*
  - ...

Item frequencies - Naive approach

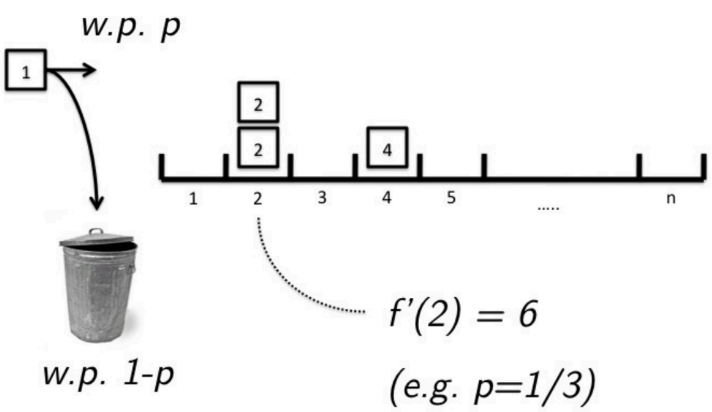
- Item frequencies
  - *n counters*
  - *Computing f (i) for all i is easy in O(n) space.*



* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

Item frequencies - sampling

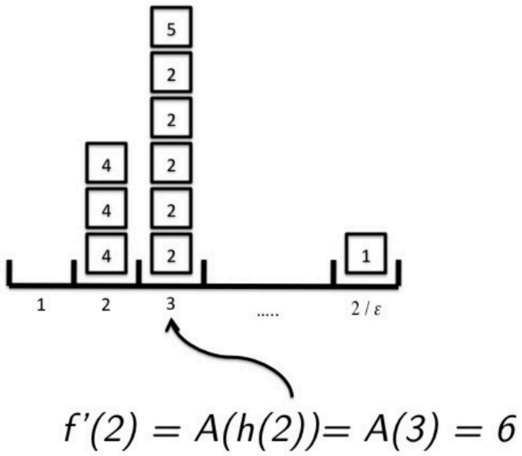
- Item frequencies - sampling
  - *We sample with probability p and estimate f(i) = 1/p A(i)*
  - *O(pN) space*



* Edo Liberty, Jelani Nelson: Streaming Data Mining - KDD 2012 tutorial on practical algorithms in mining streaming data.

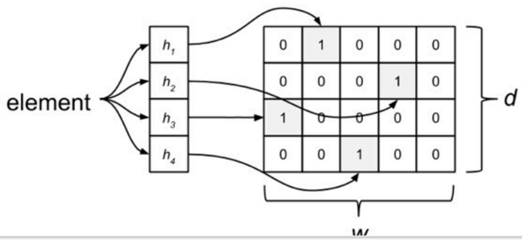
Item frequencies - Count-Min Sketch

- Count-min sketch
  - *limited amount of counters*
  - *Items are counted in buckets according to a hash function*



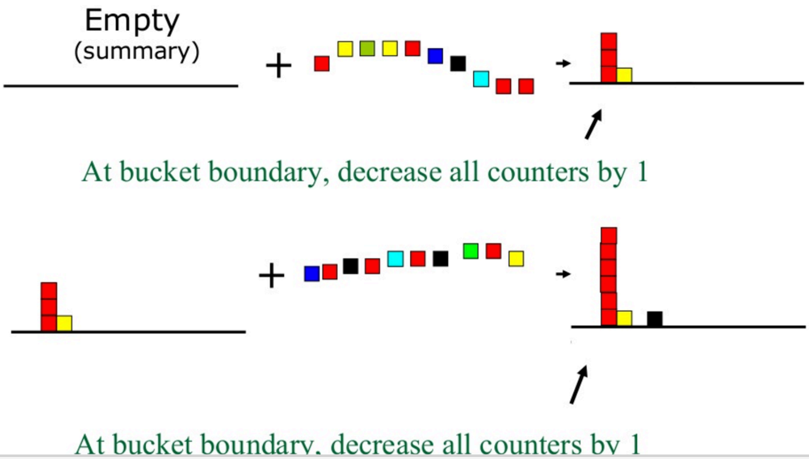
Item frequencies - Count-Min Sketch

- Count-min sketch
  - *approximate frequencies in sublinear space*
  - *matrix with w columns and d rows - initialized to zeros*
  - *each row has a hash function*
  - *When elements arrive*
    - *→ hash for each row*
    - *→ increment each counter by 1*
  - *freq(element) = min counter value*
- Minimum value
  - *possibility for collision for elements*
  - *may overestimate*



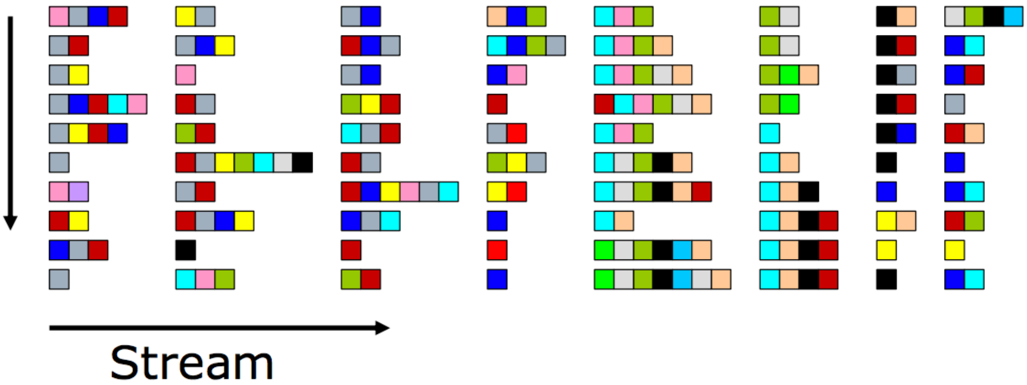
Identifying frequent items - Lossy Counting

- Lossy Counting
  - *Application in the Frequent Pattern Mining*
    - *→ Association Rules*
  - *Approach*
    - *→ Incoming stream is divided into buckets of size w*
    - *→ Compute frequencies according to the bucket*
    - *→ At bucket boundary, decrease all counters by 1*



Finding Frequent Itemsets

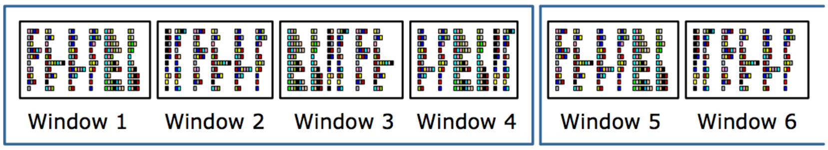
- Approach
  - *load as many buckets as possible to the main memory*
  - *delete entry if the updated frequency is less than bucket number*
  - *decrease the frequency by number of buckets*



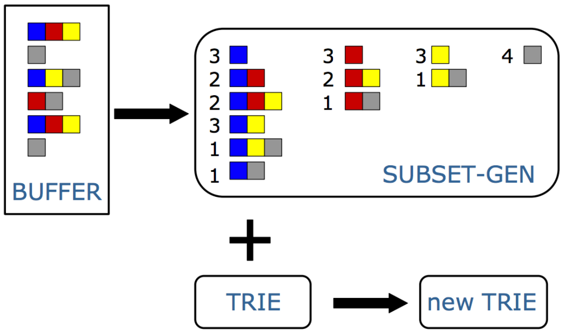
* Mining Data Streams: https://www.slideshare.net/Krish_ver2/51-mining-data-streams.

Finding Frequent Itemsets (cont.)

- *Load as many buckets as possible to the main memory*



- *Algorithm*



* Frequency Counts over Data Streams: https://www.cse.ust.hk/vldb2002/VLDB2002-proceedings/slides/S10P03slides.pdf.

Count Distinct Items

- Flajolet–Martin algorithm
  - *probabilistic computing*
  - *refined in LogLog and HyperLogLog*
    - → *Billions of distinct values in 1.5KB of RAM with 2% relative error*
- The main idea - Cardinality estimation
  - *Algorithm*
    - → $n = 0$
    - → *For each input item*
      - → *hash item into i bit string*
      - → *count trailing zeros in the bit string*
      - → *if the count* $> n$
        - → $n = count$
    - → *Cardinality* $= 2^n$
- Assumptions
  - *a good hash function*
  - *in random data, a sequence of* n *zero bits will occur once in every* 2^n *elements, on average*

Count Distinct Items (cont.)

- Flajolet–Martin example
  - ... 0 1 0 1 0 1 0 1 0 0 1 0 0 0
  - P(n trailing zeros) = $(1/2)^n$
  - Number of seen hashes = 2^n
    → ... {000, 001, 010, 011, 100, 101, 110, 111}

id	H(id)	trailing zeros	max zeros
A	0111001001	0	0
B	1000110100	2	2
C	1101111000	3	3
D	1001010110	1	3
E	1101011100	2	3

- Assumptions
  - if we had seen 2 distinct items, we would expect 1 trailing zero
  - if we had seen 4 distinct items, we would expect 2 trailing zero
  - reasonable estimation is 2^n
- Improvements
  - more independent hashes or prefix and subhashes
  - compute mean values

Count Distinct Items (cont.)

```python
# https://jeremykun.com/2016/01/04/hashing-to-estimate-the-size-of-a-stream/
import random
def randomHash(modulus):
    a, b = random.randint(0,modulus-1), random.randint(0,modulus-1)
    def f(x):
        return (a*x + b) % modulus
    return f

def average(L):
    return sum(L) / len(L)

def numDistinctElements(stream, numParallelHashes=10):
    modulus = 2**20
    hashes = [randomHash(modulus) for _ in range(numParallelHashes)]
    minima = [modulus] * numParallelHashes
    currentEstimate = 0
    for i in stream:
        hashValues = [h(i) for h in hashes]
        for i, newValue in enumerate(hashValues):
            if newValue < minima[i]:
                minima[i] = newValue

    currentEstimate = modulus / average(minima)
    return currentEstimate

S = [random.randint(1,2**20) for i in range(10000)]
for k in (10,50,100):
    print(numDistinctElements(S, k))

# 9110.130321459601
# 9463.682310469314
# 10447.105708877154
```

Computing Moving Average

- Sliding Window Approaches

```
import random
import numpy as np

full = []
queue1 = []
size = 3

def add1(value):
    if len(queue1)<size:
        queue1.append(value)
    else:
        del queue1[0]
        queue1.append(value)
    return sum(queue1)/len(queue1)

for i in range(10):
    a = random.randint(0,100)
    full.append(a)
    print("add({}) - moving={} - total={}".format(a, add1(a), np.mean(full)))
```

```
add(83) - moving=83.0 - total=83.0
add(53) - moving=68.0 - total=68.0
add(28) - moving=54.666666666666664 - total=54.666666666666664
add(18) - moving=33.0 - total=45.5
add(42) - moving=29.333333333333332 - total=44.8
add(60) - moving=40.0 - total=47.333333333333336
add(39) - moving=47.0 - total=46.142857142857146
add(37) - moving=45.333333333333336 - total=45.0
add(11) - moving=29.0 - total=41.22222222222222
add(12) - moving=20.0 - total=38.3
```


Overview

- Introduction
- Stream Data Processing
- Approaches
- Technologies

Technologies

- Stream Processing Systems
  - *can handle a nearly unlimited amount of data, but they only process one (true stream processing) or very few (micro-batch processing) items at a time*
- Many existing technologies
  - *Apache Storm*
    → stream processing framework, extremely low latency, near real-time processing
  - *Apache Samza*
    → tightly tied to the Apache Kafka messaging system
  - *Apache Spark*
    → batch processing framework with stream processing capabilities
  - *Apache Flink*
    → stream processing framework that can also handle batch tasks
  - ...

Apache Spark Example

- Docker Jupyter Notebook Python, Spark, Mesos Stack

```
docker pull jupyter/pyspark-notebook
docker run -it --rm --add-host=parent-host:10.0.0.10 -p 8880:8880 -v $PWD:/home/jovyan/work ju
```

- Processing stream

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
lines = ssc.socketTextStream("parent-host", 9999)
words = lines.flatMap(lambda line: line.split(" "))
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
wordCounts.pprint()
ssc.start()             # Start the computation
ssc.awaitTermination()  # Wait for the computation to terminate
```

- Generate stream

```
nc -lk 9999
```

- Outputs

```
(hello,1)
(world,1)
```