Mining Massive Databases

Mining Data Streams

Diego Saez-Trumper

Data Science - UCU

Mining from Data Streams

- **Different from databases:**
  - We see each data point just once
  - We cannot not save data in *active storage.*
- **Involves summarization:**
  - Sample
  - Filter
  - Buffer

Examples

- Social media streams:
  - Twitter stream
  - Wikipedia editions stream (visualization)
- Server logs
  - Http logs
  - Login systems
- Transactions
  - Credit cards
  - Bike sharing systems
- User sessions within a platform/apps
- Sensors
  - Weather, earthquakes, etc

Stream Queries

- Standing queries
  - For each data point, we produce an output indepent from other data points in the stream.
    - Example: Consider a stream of tweets. For each tweet determine in which language is written.
- Ad-hoc queries
  - Requires contextual information.
    - Example: Given a stream of tweets, determine which is the most active user in last hour.

Considerations about Streams

- When the stream is big/fast enough, approximation methods might useful:
  - Important: We should know the % of confidence of our approximation.
- Even for slow streams, we should be aware of space limits, and define summarization methods.

Sampling Streams

- Example: You want to get distribution of the number of queries that (logged) users does in search engine.
  - Given that you have too many queries, you are asked to sample the ~1% of your data.

How to create the sample?

“if you’re doing a software interview that involves some algorithms problem that seems hard, your best bet is to use hash functions.”

Sampling Streams

- Our stream consists in tuples with n components.
- A subset of components are the key values.
- To get sample size of a/b we hash the key(s) value(s) to and accept the sample if the hash value is less than a.

Filtering Streams

- In this case we already know a set of values that we want accept, but they are too big to keep in memory.
  - Example: You have one billion of allowed e-mail addresses, and you want to reject (or double check) all the e-mails not coming from those address.

How to create the filter?

The Bloom Filter

- A Bloom Filter consists of:
  - An Array of n bits, initially at all 0's
  - A collection of hash functions h1,h2…,hk
  - A set of S of m key values

- Then proceed like follows:
  - Take each key value in S and hash using all k hash functions.
  - Set to 1 each bit that is $h_i(K)$ for same hash function and hash value.
  - To test data, apply the hash function to every new data point in the stream, and if is 1 for all values in the bit array, let it pass the filter.

Does this procedure warranty that all allowed element passes?
Does this procedure warranty that all not-allowed elements are filtered?

Code

Bloom Filtering Analysis

- What is the probability of false positives?
- Considering the target/darts analogy, suppose that we have *Y* darts and *X* targets with equal probability.
  - The probability of a given dart not hit a given target is $\frac{x-1}{x}$
  - The probability of none of y darts will hit a given a tart is $\frac{(x-1)^y}{x}$
  - rewritten $(1-\frac{1}{x})^{x(y/x)}$
  - That can be approximated to $e^{-y/x}$, and the probability of hit the that target $1-e^{-y/x}$

- Example: Considering the one billion of emails ($y=10^9$) and 8 billions of bytes of memory (8G, $x=8\cdot10^9$), the probability of spam email passing the filter is $1-e^{-1/8} \sim 0.1175$

Pyspark Streaming
Hello word (count)

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

# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
lines = ssc.socketTextStream("localhost", 9999)
words = lines.flatMap(lambda line: line.split(" "))

# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.pprint()

ssc.start()             # Start the computation
ssc.awaitTermination()  # Wait for the computation to terminate

#run nc -lk  9999  to create a server
```

Challenge I: Create a bloom filter with pyspark

Counting Distinct Elements in a Stream

- Consider a stream of elements big enough to don't fit in memory. For example, all the IP address searching on Google. You want to estimate the total number of **distinct** elements in the stream.
- When data is big enough, you can not keep a list, hash, or tree of the elements already seen.
- It is possible to estimate the number of distinct elements by hashing the elements of the universal set to a bit-string sufficiently long.
- **The length of the bit-string must be sufficient that there are more possible result of the hash function than elements in the universal set.**
- The Flajolet-Martin algorithm is based on the idea that as we see more "unusual" hash-values, we will have more elements in the set.
- The "*unusualness*" that this algorithm exploit is the number of 0's at the end of the hash.
- There other characteristics of the hash-values that can be exploited
- Some implementations for the counting-distinct elements problem can be found here:
  - https://jeremykun.com/2016/01/04/hashing-to-estimate-the-size-of-a-stream/
  - http://blog.notdot.net/2012/09/Dam-Cool-Algorithms-Cardinality-Estimation

Flajolet-Martin algorithm

- Pick a hash function h that maps each of the n element to at least $\log_2 n$ bits
- For each stream element a, let r(a) be the number of trainling 0's in h(a)
- Record R= the maximum(r) seen
- Estimate = 2^R

Explaining video

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)

    yield currentEstimate

S = [random.randint(1,2**20) for _ in range(10000)]
print(len(set(S))
for k in range(10,301,10):
  for est in numDistinctElements(S, k):
    pass
  print(abs(est))
```

Google Colab Implementation

Challenge II: Use this Fajolet-Martin implementation to count the number of different Hosts in the Airbnb Data Set

Other problems for Streams

- Some interesting problems are:
  - Counting the numbers of ones in a stream of bits of length N, in given (changing) window of size K.
  - Counting the most popular elements in a sliding window.
  - Time-weighted count of the most popular elements in a sliding window.
- Suggested reading:
  - Chapter 4: Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2014.

Streams in python

- sockets (built-in)
- Tornado & twisted:
  - Websocket communication
  - TCP/UDP
  - etc
- PyOSC (for sensors)
- ws4py (Websockets)
  - Websockets
- socketio
- sseclient (server-side events)
- PySpark Streaming
  - Receive streams and put spark context
  - Usually used with **Kafka**

Challenge II: Counting number of different users in Wikipedia Stream Data

- Consider all the changes than in the wikipedia as stream.
  - Check here: https://wikitech.wikimedia.org/wiki/RCStream
  - Working code here.
- Each action is received in json format.
- Considering all the changes
  - [change = json.loads(event.data)]
  - Estimate the number of different (users,wiki) in a stream of 5000 change events.

Questions

?

Final Project: Detecting bots in wikipedia streaming data

- Consider all the changes done in the wikipedia as stream.
  - Check here: https://wikitech.wikimedia.org/wiki/RCStream
- Each action is received in json format.
- Data is full of *bots*. There is a flag were programmers can indicate that an actions has been done by a bot.
- Using this information as ground truth, develop a system able to classify users as bot or human.
- **Constrain:**
  - **You need to sample, and just use the 20% of the data stream.**
  - **Don't use username.**
  - **Use at least 40K edits**
- Describe the distribution of edits per humans and bots.
- Finally, train a Bloom Filter that filter out bots from the stream.
  - Find the correct parameters for the bloom filter having an error around 10% .
- If you want to have a 100% you need to do this:
    - Make your system to work with Spark Streaming (5%)

Mid-term: Homework

- Repeat the LSH experiment for the full **Barcelona** dataset, and develop and spark boosted methodology (ParagramGridBuilder) to have an efficient parameter tuning process.

- Do the same with the field “Title” with the April-2019 top-1000 Articles in Ukrainian Wikipedia. You can use the Wikimedia Pageview Tool or the Pageviews API
  - Your report should include:
    - The accuracy of your model with at least 4 combinations of parameters
    - The computation time spent in the parameter tuning procedure (specify also the characteristics of the machine(s) that you have used)
    - What happen if we tune the parameters using Airbnb and the run with those parameters in The Wikipedia Dataset? What is the difference in accuracy with the parameters tuned directly in Wikipedia data?

Evaluation

- Homework I:
  - 30%
  - November 10th, 2024
- Final Project:
  - 70%
  - November 19th, 2024
- Format:
  - Jupyter-Notebook that works on this docker.
  - Or self-standing Google Colab
- Groups: 2/3 Students
- Oral Presentation: November 21th