Fuzzy Counting in Stream Data using Probabilistic Data Structures

A Probabilistic Alternative to Bloom and Cuckoo Filters

B12902036吳述宇、B12201033吳竣凱、B10902120張耕綸、B10505001謝博仲

Outline

- Baseline
  - Fuzzy Counting in Streaming Setting
  - Algorithms
- HyperLogLog Improvement
  - Motivation and Challenges
  - Algorithms
- Experiment Analysis
  - Parameters and Metrics
- Autocorrection Library
- Future Work

Local Differentially Private Fuzzy Counting in Stream Data using Probabilistic Data Structures

Dinusha Vatsalan, Raghav Bhaskar, and Mohamed Ali Kaafar

- Exact Matching ⟶ Fuzzy Counting + Privacy
- Great Time and Space Efficiency with Accuracy
- New LDP Method — Self-made Bloom Filters

Differential Privacy

- A randomized algorithm A satisfies ε-Differential Privacy if for two neighboring datasets D and D’ and any output set.

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(D') \in S]$$

- Smaller ε means stronger privacy.
- If someone gets inserted or deleted from the dataset, the overall statistical outputs will not change.

Baseline

- Problem : How to maintain both accuracy and privacy for fuzzy counting in streaming environment?
  - Sensitive data is everywhere.

- Fuzzy Counting
  - A probabilistic estimation technique that counts approximately matching strings, allowing tolerance for typos, misspellings, or semantic variations.
  - Limited Memory, Accuracy, Privacy

Fuzzy Counting in Streaming Setting

- In the streaming model, we observe a sequence of items one by one with limited memory.
- The goal is to estimate the number of approximate matches to a given query string at any time.
  - Fuzziness
  - Space Efficiency
  - Low Latency

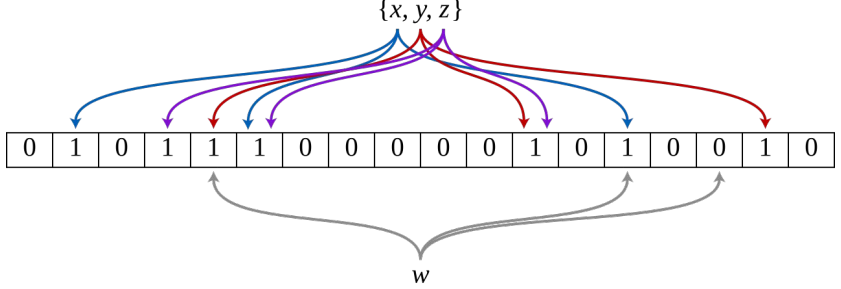
CMS and RAPPOR

CM Sketch:

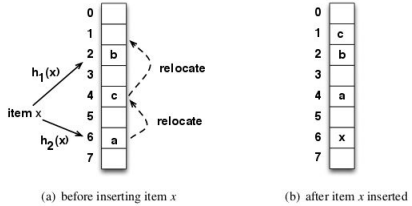
1. Let $h_1, h_2, ..., h_k$ be k hash functions from $[U]$ to $[m]$.
2. Create k arrays $A_1, A_2, ..., A_k$, each of size m.
3. For each element x:
   - For each j $(1 \leq j \leq k)$, increment $A_j[h(x)]$ by 1.
4. For each key $x \in [U]$, compute:
   - $\hat{f}_x := \min_j A_j[h(x)]$
5. Return $\max_{x \in [U]} \hat{f}_x$.

- Count-Min Sketch(CMS)
  - Linear-time update
  - Proven bounds on overestimation
  - Only support exact matching and no builtin privacy
- RAPPOR
  - Local Differentially Privacy
  - Only support exact value frequency estimation
  - No fuzziness
- We need a new approach for **Fuzziness + Efficiency + Privacy**!

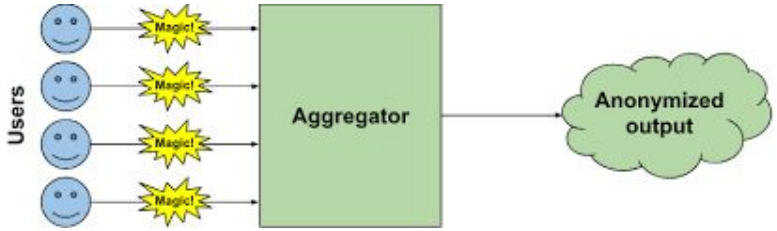
Preliminaries



- Bloom Filter
  - Encode input strings into fixed-length segment fingerprints, enabling efficient fuzzy matching and segment-based querying as a preprocessing step.
- Cuckoo Filter
  - Store and query the segment fingerprints generated by the Bloom Filter, providing efficient counting and membership tests with low false positive rates and dynamic capacity for real-time fuzzy counting.
- Similarity Threshold(st)
  - The minimum required overlap between the query string and candidate strings' segments.



Algorithms



- Phase 1 : Fuzzy Encoding
  - Divide every input string to several substrings(q-gram)
  - Use locality hashing for approximate matching
- Phase 2 : Local Randomization(LDP)
  - Add noises locally to the data
    - However, it generates self-made Bloom Filter instead of directly adding noises to the bits.
  - Generalized Randomized Response
  - Ensuring DP before entering aggregator
- Phase 3 : Aggregation & Decoding
  - Server collects noisy sketches from multiple users
  - Return the count according to the matched segments and the similarity threshold

Performance

- Insertion Time
  - Higher but less than an order of magnitude than RAPPOR and CMS
- Query Latency
  - Less than 1~1.5 order of magnitude in seconds
- False Positive Rate
  - Near 0 even above 1000000 records
  - CMS will reach 1 near 10000 records, and RAPPOR will reach 1 near 500000 records.
- Variance
  - In both exact counting and fuzzy counting setting, Cuckoo Filter outperforms both RAPPOR and CMS by 2~3 orders of magnitude.

HyperLogLog

HyperLogLog Improvement

In the future, we aim to study noise addition according to "local sensitivity" of the bucket in the Cuckoo filter in order to add noise based on the set of items the bucket has seen so far which could improve the utility and efficiency for a given privacy budget. The proposed differential privacy algorithm requires that clients need to keep track of the items reported to the server in order to avoid reporting the same item multiple times (i.e. only unique items are reported to the server). In the future, we aim to address the local storage/memory efficiency aspect by using efficient probabilistic data structures, such as HyperLogLog.

Source: [VBK22]

- We wanted fuzziness because it saves data... Data sketching?
- HLL is a very space-efficient distinct objects counter
- HLL is quite fast with insertion and estimation
- HLL supports merging and stays small → effective for large amount of users
- HLL doesn't store full inputs → Leave less footprints (obfuscation), so privacy could be achieved easier later on

TL;DR: HLL wasn't designed for language modeling, but if each q-gram acts like a signal, HLL can help us estimate which signals are called by the most users

HyperLogLog Challenges

- The baseline BloomCuckoo (BC) focused on fuzzy encoding, but HLL only considers exactness
- Reframe our question from “fuzzy similarity” to “q-gram popularity scoring” (Seems reasonable, right?)
- BC reconstructs words via matched fingerprints, HLL does not have so built-in… Doesn’t HLL usually seem to be thought of as unable to trace back to the original source?
    - We explicitly stored a map of “q-gram → candidate words”, and ranked by HLL-estimated frequency, then output the most likely word(s), which likely led to more memory usage

TL;DR: This was not simply a small change to the baseline algorithm, it is a structurally different point of view and implementation of autocorrection and HLL usage

HyperLogLog Algorithms (No Privacy Version)

BC

- Phase 1 : Fuzzy Encoding
  - Divide every input string to several substrings(q-gram)
  - Use locality hashing for approximate matching
- Phase 3 : Aggregation & Decoding
  - Server collect noisy sketches from multiple users

HLL

- Phase 1: Extract q-gram Encoding
  - Split every string into q-grams
  - Insert into (user, q-gram) HLL sketch (eg “ap_user1” if user1 types “apple”)
- Phase 3: Candidate Scoring & Output
  - Maintain a q-gram → candidate words map
  - For each query
  - Tally how many q-grams it shares with each candidate
    - Score = Estimated overlap count by HLL frequency

Experiment Analysis

- Dataset
  - Wikipedia List of common misspelling/For machine & kaggle from bittlingmayer Spelling Corrector - extract 528 typo and correction
  - Google-10000-english github - 9894 words
- Metrics
  - Accuracy
  - Execution Time
  - Memory Usage

Result

- Accuracy
  - Baseline 76.52% / HLL 72.16% (database 528 words)
  - Baseline 66.66% / HLL 52.83% (database 10343 words)
- Execution Time
  - Baseline 5.472s / HLL 0.562s (database 10343 words)
- Memory Usage
  - HLL use 7.89% less memory than baseline (database 10343 words)

AutoCorrection Library

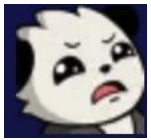
- Create HLL library let user can import our library using

```
!pip install hll_py

from hll_py import AutoCorrector
```

```
history_input = [
    ("u1", "apple"),
    ("u2", "banana"),
    ("u3", "grape"),
    ("u4", "orange"),
    ("u5", "banana"),
]
queries = ["applle", "banana", "banan", "orenge", "grap", "pineapple"]
```

```
  -> Suggested correction: apple
------------------------------

  -> Suggested correction: banana
------------------------------

  -> Suggested correction: banana
------------------------------

  -> Suggested correction: orange
------------------------------

  -> Suggested correction: grape
------------------------------

  -> Suggested correction: apple
------------------------------
```

Future Work 0: Was HyperLogLog given a fair chance?



還想說HLL
不錯的你 :(

```
 titalate -> matched segments: ti(1.0326163382011386) it(1.0326163382011386) ta(1.0326163382011386)
al(1.0326163382011386) la(1.0326163382011386) at(1.0326163382011386) te(1.0326163382011386)
 -> Estimated users: 1.0326163382011386
 -> Suggested correction: installation
```

- BC corrected “titalate” to titillate, whereas HLL... installation?!
- BC uses fuzzy matching via segment overlap of fingerprints, but does not rely directly on frequency
  - Picks candidate with many shared segment encodings even if not exact
- HLL is a probabilistic counter: estimates how many users typed each exact q-gram without fuzziness, and its algorithm relies on segment-level frequency
  - Suggests words that have high segment frequency, even if unrelated
- HLL can only be tested fairly if given a database of **fairly weighted words**, or else it is very prone to overfitting

Future Work 1: HyperLogLog LDP (A Sketch)

- “HLL is so much faster than BC! Why is it not mainstream?” → Privacy
- We can try privacy if we preserve HLL merging
- Idea 1: Noise on HLL register bits (like how we flip bits in BC)
  - Technical detail: HLL checks the number of leading zeros and saves the maximum number, so we can flip high registers with low probability and low registers with high probability (More detail needs to be proved, hence why this is a sketch)
- Idea 2: Use local hashing of user identity to preserve anonymity but still get per-user uniqueness
- Idea 3: Each user only reports a random subset of their q-grams?

Challenges:

- May have identity collisions (Need to ensure a good enough hashing)
- Privacy-utility tradeoff: too much noise breaks accuracy, too little fails LDP

Future Work 2: HyperLogLog x BloomCuckoo (A Sketch)

- ... Alternatively, get LDP with BC but preserve the benefits of HLL
- BC relies on fingerprint counting of each bucket, which takes a lot of memory
- In this case, HLL can be used instead within each bucket

Challenges:

- No reverse mapping with HLL, cannot tell what input led to a change in bits, i.e. cannot delete words from a dictionary
- HLL's precision may decrease with top-k words since it uses approximate counting

Takeaway

- HLL is space-efficient, fast, and easily mergeable
- HLL can be a promising direction towards autocorrection in a new light, which suggests a new possible usage of HLL with language processing
- Data sketching is a plausible angle to handle some NLP-adjacent problems
- Future work: LDP can either be done on HLL natively, or we can fuse BC and HLL

References

[Fla+07] Philippe Flajolet et al. “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”. In: *DMTCS Proceedings*. Ed. by Philippe Jacquet. Vol. DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07). DMTCS Proceedings. Juan les Pins, France: Discrete Mathematics and Theoretical Computer Science, June 2007, pp. 137–156. DOI: 10.46298/dmtcs.3545. URL: https://inria.hal.science/hal-00406166.

[VBK22] Dinusha Vatsalan, Raghav Bhaskar, and Mohamed Ali Kaafar. “Local Differentially Private Fuzzy Counting in Stream Data Using Probabilistic Data Structures”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.8 (2022), pp. 8185–8198. DOI: 10.1109/TKDE.2022.3198478.