DSC 190

DATA STRUCTURES & ALGORITHMS

Lecture 18 | Part 1

The Count-Min Sketch

Last Time: Membership Queries

- You've collected 1 billion tweets.[1]

- **Goal**: given the text of a new tweet, is it already in the data set?

- Data set is too large to fit into memory.

- Our solution: **Bloom filters**.

[1] This is about two days of activity.

Today: Frequencies

- You’ve collected 1 billion tweets.
- **Goal**: given the text of a tweet, how many times have we seen it?
- Data set is too large to fit into memory.
- Today’s solution: the **Count-Min Sketch**.

Frequency Counts

- **Given:** a collection $X = \{x_1, x_2, \ldots, x_n\}$.
- **Support:**
  - `.count(x)`: Number of times `x` appears.
  - `.increment(x)`: Increment count of `x`

Simple Solution

- Use hash tables: dictionary of counts.

```
class SetCounts:

    def __init__(self):
        self.counts = {}

    def increment(self, x):
        if x not in self.counts:
            self.counts[x] = 1
        else:
            self.counts[x] += 1

    def count(self, x):
        try:
            return self.counts[x]
        except KeyError:
            return 0
```

Problem: Memory Usage

- Requires storing the keys.
- Example: store approximately 1 billion tweets (100 GB).
- Can’t fit the dictionary in memory.

A Fix

- Why do we store all of the keys?
- To resolve collisions.
- What if we ignore collisions?

Hashing Into Counters

0	1	2	3	4	5	6	7	8	9

s	hash(s)
"surf"	3
"sand"	8
"data"	5
"sun"	1
"beach"	5

```
"data"
"surf"
"sand"
"surf"
"surf"
"beach"
"data"
"beach"
"surf"
"sun"
```

- Use a size c ($c \ll n$) array of integers (counts).
- `.increment(x): arr[hash(x)] += 1`
- `.count(x): return arr[hash(x)]`

Hashing Into Counters

0	1	2	3	4	5	6	7	8	9

s	hash(s)
"surf"	3
"sand"	8
"data"	5
"sun"	1
"beach"	5

"data"
"surf"
"sand"
"surf"
"surf"
"beach"
"data"
"beach"
"surf"
"sun"

- Use a size c ($c \ll n$) array of integers (counts).
- .increment(x):
  `arr[hash(x)] += 1`
- .count(x):
  `return arr[hash(x)]`
- Can be **wrong**!

Biased Estimate

- The count returned from this approach is **biased high**.
- Can we do better?
- **Idea**: multiple hashing. Perform previous k times.
- This is the **count-min sketch**.

Count-Min Sketch

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

s	hash_1(s)	hash_2(s)
"surf"	3	7
"sand"	8	7
"data"	5	4
"sun"	1	9
"beach"	5	6

"data"
"surf"
"sand"
"surf"
"surf"
"beach"
"data"
"beach"
"surf"
"sun"

- Use k arrays of counts, each with own independent hash functions.
- `.increment(x)`: Set
  `arr_1[hash_1(x)] += 1`,
  `arr_2[hash_2(x)] += 1`,
  ...,
  `arr_k[hash_k(x)] += 1`.

Count-Min Sketch

0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9

s	hash_1(s)	hash_2(s)
"surf"	3	7
"sand"	8	7
"data"	5	4
"sun"	1	9
"beach"	5	6

"data"
"surf"
"sand"
"surf"
"surf"
"beach"
"data"
"beach"
"surf"
"sun"

- Use *k* arrays of counts, each with own independent hash functions.
- `.count(x)`: Return the **minimum** of `arr_1[hash_1(x)]`, `arr_2[hash_2(x)]`, ..., `arr_k[hash_k(x)]`.

Returning the Minimum Count

- The count is still biased high.
- But by returning the minimum, bias is reduced.

Memory Usage

- Each counter cell stores an integer (64 bits).
- Total size:
$$64 \times c \cdot k \text{ bits}$$
- c and k should be chosen to match prescribed level of error.

DSC 190

DATA STRUCTURES & ALGORITHMS

Lecture 18 | Part 2

Designing a Count-Min Sketch

Error Rate

- Count-min sketch is a probabilistic data structure.
  - Returns the wrong answer sometimes.
- How wrong is it, probably?
- And how does this depend on c and k?

Notation

- We see n items, record frequencies in count-min sketch.
- For any item x, let f_x be its true frequency.
- $\hat{f}_x^{(i)} \equiv$ `arr_i[hash_i(x)]` is estimated frequency of x according to row i. $\hat{f}_x$ is aggregate estimate: $\hat{f}_x = \min_i \hat{f}_x^{(i)}$.
- Note: $\hat{f}_x^{(i)} \geq f_x$

Absolute and Relative Error

- Absolute error: $\hat{f}_x - f_x$
  - This will grow as collection size $n \to \infty$.
- Relative error: $(\hat{f}_x - f_x)/f_x$
  - We're more interested in this. Want it to be small.
  - If $f_x = \Theta(n)$, we want:

$$(\hat{f}_x - f_x)/n < \varepsilon \implies \hat{f}_x - f_x < \varepsilon n$$

Analyses

- We'll first look at the expected value of the estimate in a single row.
- Then, we'll compute the probability that the aggregate estimate is much larger than the true value.

Expected Value

- Fix an object, x, and a row i.

$$\mathbb{E}[\hat{f}_x^{(i)}] = \text{expected count in } x\text{'s bin}$$

$$= f_x + \mathbb{E}[\text{tot. frequency of colliding items } y \neq x]$$

$$= f_x + \sum_{y \neq x} f_y \cdot \mathbb{P}(\texttt{hash(y) == hash(x)})$$

$$= f_x + \frac{1}{c} \sum_{y \neq x} f_y \leq f_x + \frac{n}{c}$$

Expected Value

- We found: $\mathbb{E}[\hat{f}_x^{(i)}] \leq f_x + \frac{n}{c}$.
- Is this good or bad?
  - Suppose $f_x = p_x n$, where $p_x \in [0, 1]$.
  - Absolute error is $\Theta(n)$.
  - But **relative** error is $\frac{1}{pc}$.
  - Independent of n!

Extreme Values

- Goal: show unlikely for $\hat{f}_x^{(i)}$ to be much larger than f_x
- Let's find α s.t. $\mathbb{P}(\hat{f}_x^{(i)} - f_x > \alpha) < 1/2$. Then:

$$\begin{aligned}\mathbb{E}[\hat{f}_x^{(i)}] &\geq f_x + \alpha \cdot P(\hat{f}_x^{(i)} - f_x > \alpha)\\ &= f_x + \alpha/2\end{aligned}$$

- We know $\mathbb{E}[\hat{f}_x^{(i)}] \leq f_x + \frac{n}{c}$, so $\alpha < 2n/c$.

Extreme Values

- We've shown that $\mathbb{P}(\hat{f}_x^{(i)} - f_x > 2n/c) < 1/2$.
- This is just for the ith row.
- Minimum is $> 2n/c$ only if *every* row is $> 2n/c$.
- Probability of this happening:

$$\prod_{i=1}^{k} \mathbb{P}(\hat{f}_x^{(i)} - f_x > 2n/c) \leq \left(\frac{1}{2}\right)^k$$

Extreme Values

- Let $\hat{f}_x$ be the aggregate estimate. We have shown:

$$\mathbb{P}(\hat{f}_x - f_x > 2n/c) < \left(\frac{1}{2}\right)^k$$

- Want $\hat{f}_x - f_x < \varepsilon$. Set $c = 2/\varepsilon$.

- To ensure that an over-estimate larger than ε occurs with probability δ, set

$$\left(\frac{1}{2}\right)^k = \delta \quad \Longrightarrow \quad k = \log_2 \frac{1}{\delta}$$

Designing a Count-Min Sketch

- Pick your ε and δ: "I want overestimates to be smaller than εn at least $1 - \delta$ percent of the time."
- Set number of buckets to $c = 2/\varepsilon$
- Set number of rows/hash functions to $k = \log_2 1/\delta$.

Example

- We have 1 billion tweets, want to count number of occurrences for each.
- Assume each tweet requires 800 bits.
- `dict`: around 100 gigabytes, assuming ≈ 1 billion unique

Example

- Instead, use a count-min sketch. Say, $\varepsilon = .001$ and $\delta = .01$.
- $c = 2/\varepsilon = 2000$
- $k = \log_2 1/\delta \approx 7$.
- Memory: $7 \times 2000 \times 64$ bits $= 112 kilobytes$

Example

- Now supposed you have 42 quadrillion tweets.
- `dict`: 4.2 exabytes
- count-min sketch: 112 kilobytes

How?

- The relative error ε of a count-min sketch does not depend on n!

- The n is "hidden" inside the relative error:

$$\hat{f}_x - f_x < \varepsilon n$$

Count-Min Sketch and Bloom Filters

- The Count-Min Sketch and Bloom Filters are both probabilistic data structures.
- Both make use of multiple hashing.
- Why does CMS take much less memory?

Less Memory

- Why does a CMS use less memory than a Bloom filter?
- The problem it is solving is easier.
- Bloom filter: big difference between seeing an element once and never seeing it.
- Count-Min sketch: essentially no difference.