Genomics Quiz

tyang27

December 2019

Markov

CPG island

Part of the genome where CG occurs particularly frequently. Plays a role in modulating gene expression. We would like to tell whether a k-mer is in an island or not.

Markov Assumption

Assumption: Probability of item at position k depends only on previous position, x_{k-1}

$$P(x) = P(x_k|x_{k-1})P(x_{k-2}|x_{k-3})\dots P(x_1)$$

Algorithm

1. Let $P(B|A)$ be the number of times AB appears divided by A?.
2. Given all CpG island substrings, for all combinations of letters, $P(x_i|x_{i-1})$.
3. Let the labels of Markov chain edges from A to B be $(P(B|A))$.
4. To get probability of inside CpG island, do calculation. Since we are multiplying probabilities, change to log to avoid underflow.

$$P(x_1)\prod_{i=2}^{n} P(x_i|x_{i-1}) \implies \sum_{i=2}^{n} \log P(x_i|x_{i-1}) + \log P(x_1)$$

5. Do the same for substrings not in CpG islands. Can either log or not.

$$\log \frac{P(x) \text{ in CpG}}{P(x) \text{ not in CpG}}$$

Sketching

Problems

- Set cardinality estimation

$$|A|$$

- Set intersection

$$|A \cap B|$$

- Set union

$$|A \cup B|$$

- Set similarity

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- Containment

$$Containment(A, B) = \frac{|A \cap B|}{|B|}$$

Hat problem

Given cards labelled 1 to N, then you pick a random subset A. Estimate $|A| = n$ using a constant number of representatives. Recall, elements in a set are unique. This relates to genomics because we hash our k-mers to get numbers, and then want to know cardinality.

MinHash

Cardinality

- Min strategy - keep track of minimum. Then, assume that this tells us the average segment size. Assume $n + 1$ intervals given by n partitions (the chosen cards).

$$min = N/(n + 1)$$

$$n = (N/min) - 1$$

- Bottom k strategy - keep track of k smallest elements, and use k-th element $mink$ to tell us average size of k segments. Good for two hat problem, e.g. cardinality of two sets under set operation.

$$mink = k * N/(n + 1)$$

$$mink/k = N/(n + 1)$$

$$n = (N * k/mink) - 1$$

- Min of k partitions strategy - take the min of k hash functions. Same result as bottom k strategy in practice.
- K-partition strategy - keep track of mins of k partitions. Good for cardinality of two lopsided sets under set operations. Used in HyperLogLog

Larger k has more of an averaging effect, better estimate, with tradeoff of more work and storage. Each representative fits in $\lceil \log U \rceil$, where U is the universe size, since we are storing hashes in binary.

Jaccard

- Union of bottom k - Combine two sketches by taking bottom k of both subsets' sketches, unioned. Then, count the number of items in both, and divide by k. This will be an estimate of fraction of cardinality of intersection divided by cardinality of union.
- Min of k partitions strategy - take the min of k hash functions. Count the number of items in both, as above. This scheme is good for comparing lopsided sets (one is large, one is small and localized).

LogLog

- Intuition 1: We don't necessarily need to care about the exact minimum, can use approximate minimum by logging. Then, re-exponentiate later, though it won't precisely minimum. This saves space.
- Intuition 2: Maximum leading zero count (LZC) of binary number roughly corresponds to the log of the minimum number. The larger the set, the smaller the min, the larger the LZC. When the set doubles, LZC increases by 1.

HyperLogLog (HLL)

- k-partition using the first $\log_2 k = p$ bits.
- Get the max LZC or log2 min of the rest of the bits for each partition. This is the HLL array.
- Re-exponentiate.
- Averaging, bias correction.

Operations

- Union - elementwise max of LZC or elementwise min of log2 both sketches will roughly correspond to minimum of union.
- Intersection - elementwise compare.

Tricks

- Vectorized and thus parallelized operations for elementwise maximum, etc.
- Change log base to maximize bits in register used.

Bloom Filters

Esp. good for containment. Hash, then set bit to 1. We can use k hash functions to better account for hash collisions. m is size of filter.

- Probability a given bit is 1: $1/m$.
- Probability a given bit is 0: $1 - 1/m$
- Probability a given bit is 0 after n insertions, k hashes: $(1 - 1/m)^{nk} \approx e^{-nk/m}$
- False positive rate: $[1 - (1 - 1/m)^{nk}]^k$

The ideal number of hash functions is $m/n * \ln 2$.

Operations

- Union - bitwise or
- Intersection - not so easy
- Containment - Set fraction of 0's to $e^{-nk/m}$, then solve for n.

Sequence bloom tree

Bitwise or of each read. Then, traverse the true, and don't go into a section the moment you see a 0.