Freie Universität Berlin

Probabilistic counting

Nano course probabilistic counting and sketching

Knut Reinert

Summer 2025

Introduction

Today

- Some ideas about estimating counts and cardinalities of sets
- Concentration bounds

Literature

- Nicola Prezza: Skript **CM0622 - Algorithms for Massive Data**, University of Venice (arxiv)
- Knut Reinert: Lecture notes Advanced Algorithms (extended version of these slides)

Concentration inequalities

Concentration inequalities provide bounds on how likely it is that a random variable deviates from some value (typically, its expected value). These will be useful in the next sections to calculate the probability of obtaining a good enough approximation with our randomized algorithms.

Markov's inequality

Suppose we know the mean $E[X]$ of a nonnegative random variable (RV) X. Markov's inequality can be used to bound the probability that a random variable takes a value larger than some positive constant.

Concentration inequalities

Lemma (Markov)

For any nonnegative RV X and $a > 0$ we have: $P(X \geq a) \leq E[X]/a$.

Proof

We prove the inequality for discrete RVs (the continuous case is similar).
$E[X] = \sum_{x=0}^{\infty} x \cdot P(X = x) \geq \sum_{x=a}^{\infty} x \cdot P(X = x) \geq a \cdot \sum_{x=a}^{\infty} P(X = x) = a \cdot P(X \geq a)$.

Example

As an example lets throw 100 fair coins and count the number of heads, then we do know that $E(X) = 50$, so the inequality tells us that $P(X \geq 90) \leq 50/90 = 0.555$. That is not very impressive. But we can do better.

Concentration inequalities

Chebychev's inequality

Chebyshev's inequality gives us a stronger bound than Markov's, provided that we know the random variable's variance. This inequality bounds the probability that the RV deviates from its mean by some fixed value.

Lemma (Chebychev)

For any $k > 0 : P(\mid X - E[X] \mid \geq k) \leq Var[X]/k^2$.

Proof

We simply apply Markov to to the RV $(X - E[X])^2$:

$$P(\mid X - E[X] \mid \geq k) = P((X - E[X])^2 \geq k^2) \leq E[(X - E[X])^2]/k^2 = Var[X]/k^2$$

Concentration inequalities

Example

So in our example of 100 fair coin throws, we do know that $Var(X) = 25$, so the inequality tells us that $P(|X - 50| \geq 40) \leq 25/1600 = 0.015625$. That is already much more impressive. But we can do even better.

Boosting Chebychev's inequality

A trick to get a better bound is to draw s values of the RV X and average out the results. Let $X' = \sum_{i=1}^{s} X_i/s$ (note: this is itself a RV) be the average of the s pairwise-independent random variables, distributed as X. Then:

Lemma (Boosted Chebyshev's inequality)

For any $k > 0$ and integer $s \geq 1 : P(|X' - E[X]| \geq k) \leq Var[X]/(s \cdot k^2)$.

Concentration inequalities

Proof

Since the X_i are i.i.d, we have $E[\sum_{i=1}^{s} X_i] = s \cdot E[X]$. For the same reason, $Var[\sum_{i=1}^{s} X_i] = s \cdot Var[X]$. Then

$$
\begin{aligned}
P(\mid X' - E[X] \mid \geq k) &= P(s\cdot \mid X' - E[X] \mid \geq s \cdot k) && (1)\\
&= P(\mid \sum_{i=1}^{s} X_i - E[\sum_{i=1}^{s} X_i] \mid \geq s \cdot k) && (2)\\
&\leq Var[\sum_{i=1}^{s} X_i]/(s \cdot k)^2 && (3)\\
&= s \cdot Var[X]/(s \cdot k)^2 && (4)\\
&= Var[X]/(s \cdot k^2) && (5)
\end{aligned}
$$

Concentration inequalities

Example

Assume now we take the average X' of 100 experiments in which we count the heads of 100 fair coin throws. Hence $s = 100$ and
$P(|X' - 50| \geq 40) \leq 25/(100 \cdot 1600) = 0.00015625$. This is now really good.

For another, smaller value of deviation we get $P(|X' - 50| \geq 2) \leq 25/(100 \cdot 4) = 0.0625$. So the probability to deviate more than 2 from 50 is only at most about 6%.

Sometimes (like in the coin throw example) we can do **even** better using Chernoff-Hoeffding inequalities.

Concentration inequalities

Chernoff-Hoeffding inequalities

Chernoff-Hoeffding's inequalities are used to bound the probability that the sum $Y = \sum_{i=1}^{n} Y_i$ of n independent identically distributed (iid) RVs Y_i exceeds by a given value its expectation. The inequalities come in two flavors: with additive error and with relative (multiplicative) error. We will prove the additive form.

The inequalities give a much stronger bound w.r.t. Markov, precisely because we know a particular property of the RV Y (i.e. it is a sum of iid. RVs). Here we study the simplified case of Bernoullian RVs.

Concentration inequalities

Chernoff-Hoeffding bound, additive form

Let $Y_1, \ldots, Y_n$ be fully independent Bernoulli random variables. Denote $Y = \sum_{i=1}^{n} Y_i$. Then, for all $t \geq 0$:

- $P(Y \geq E[Y] + t) \leq e^{-\frac{t^2}{2n}}$
- $P(Y \leq E[Y] - t) \leq e^{-\frac{t^2}{2n}}$
- $P(|Y - E[Y]| \geq t) \leq 2 \cdot e^{-\frac{t^2}{2n}}$

Equivalently, let $Y' = Y/n$ (i.e. the average of all Y_i) be an estimator for p. Then, for all $0 < \epsilon < 1$:

$$P(|Y' - p| \geq \epsilon) \leq 2 \cdot e^{-\frac{\epsilon^2}{2n}}$$

Concentration inequalities

Example

Before we prove the inequality, let's look at our example. 100 fair coin throws are exactly Bernoulli distributed RVs that are iid. Then it holds (without boosting) that

$$P(|X - 50| \geq 40) \leq 2 \cdot e^{-\frac{40^2}{200}} = 0.00067.$$

Compare this to the 5/9 of Markov and the 0.015625 of Chebychev.

Concentration inequalities

Chernoff-Hoeffding bound, multiplicative form

If $E[Y]$ is small, a bound on the relative error is often more useful:

Lemma (Chernoff-Hoeffding bound, multiplicative form)

Let $Y_1, \ldots, Y_n$ be fully independent Bernoulli random variables. Denote $Y = \sum_{i=1}^{n} Y_i$ and $\mu = E[Y] = np$. Then, for all $t \geq 0$:

- $P(Y \geq (1+\epsilon)\mu) \leq e^{-\mu\epsilon^2/3}$
- $P(Y \leq (1-\epsilon)\mu) \leq e^{-\mu\epsilon^2/3}$
- $P(|Y-\mu| \geq \epsilon\mu) \leq 2 \cdot e^{-\mu\epsilon^2/3}$

A useful Lemma

As an example for Chernoff-Hoeffding we turn now to counting (up to n) and state a useful lemma.

Taking the median of estimates

Let us denote with $\Theta_1, \ldots, \Theta_t$ t instances of an estimator for the expected value. The median $\Theta^+ = \mathit{median}\{\Theta_1, \ldots, \Theta_t\}$ of the t instances. We now show that the median is indeed correct (has relative error bounded by ϵ) with high probability, and we compute the total space usage (i.e. number of registers) required to guarantee relative error ϵ with probability at least $1 - \delta$, for any choice of ϵ and δ.

Lemma

Let Θ^+ be an estimator such that $E[\Theta^+] = n$ and $P(|\Theta^+ - n| > n \cdot \epsilon) \leq 1/3$. For any desired failure probability δ, draw $t = 72 ln(1/\delta)$ instances of the estimator and define $\Theta^{++} = \mathit{median}\{\Theta_1^+, \ldots, \Theta_t^+\}$. Then:

$$P(|\Theta^{++} - n| > n \cdot \epsilon) \leq \delta$$

A useful Lemma

Example

You want to determine whether a coin is fair by throwing it 100000 times. You would tolerate a deviation of $\epsilon = 1\%$ from the ideal value of 50000 heads to call it fair. How often do you have to repeat the 100000 coin throws to get enough estimates such that your error probability is less than $\delta = 1\%$?

The answer is: It is $72 \cdot \ln(100) \approx 332$ times. We only have to show that $P(|\Theta^+ - n| > n \cdot \epsilon) \leq 1/3$. An application of Chernoff-Hoeffding in additive form yields $P(|\Theta^+ - 50000| \geq 500) \leq 2 \cdot e^{-500^2/200000} = 0.28$. Hence, we have a less than 1% error chance that the median estimator would deviate by more than 1%.

Notes on hashing

Let $\mathcal{H}$ be a family of functions $h : [1, n] \to \{x \in R | 0 \leq x \leq 1\}$ mapping the integers $[1, n]$ to the real interval $\{x \in R | 0 \leq x \leq 1\}$. To simplify notation, in these notes, we will denote the codomain $\{x \in R | 0 \leq x \leq 1\}$ with $[0, 1]$ (not to be confused with the interval of integers [1, n] of the domain). The integer/real nature of the set [a, b] will always be clear from the context.

We say that $\mathcal{H}$ is k-wise independent iff, for a uniformly-chosen $h \in \mathcal{H}, (h(x_1), \ldots, h(x)_k))$ is uniform in $[0, 1]^k$ for any choice of distinct $x_1, \ldots, x_k$. It is impossible to algorithmically draw (and store) a uniform function $h : [1, n] \to [0, 1]$, since the interval $[0, 1]$ contains infinitely-many numbers. However, we can use an approximation with any desired degree of precision using pairwise-independent discrete hash function $h' : [1, n] \to [0, M]$.

Counting distinct elements of a set

In this section we consider the following problem. Suppose we observe a stream of m integers $x_1, x_2, x_3, \ldots, x_m$ (arriving one at a time) from the interval $x_i \in [1, n]$ and we want to count the number of **distinct** integers in the stream, i.e. $d = |\{x_1, x_2, \ldots, x_m\}|$. We cannot afford to use too much memory (and m and d are very large — typically in the order of thousands of billions).

This models **online** streaming of elements, but also the use case that you want to count the number of distinct elements in large files while only reading them **once**.

Example

You might want to count the number of different 30-mers in RefSeq. Refseq contains about 1000 *Gbases*. There exist $n = 4^{30} = 1.152.921.504.606.846.976$ different 30-mers. So potentially, d could could be as high as 10^{12} if all the 30-mers are different.

Counting distinct elements of a set

Naive solution

A first naive solution to the count-distinct problem is to keep a bitvector $B[1, n]$ of n bits, initialized with all 0's. Then it is sufficient to set $B[x_i] = 1$ for each element x_i of the stream. Finally, we count the number of 1's in the bitvector. If n is very large (like in typical applications), this solution uses too much space. In the next slides we will see how to compute an approximate answer within logarithmic space.

Idealized Flajolet-Martin's algorithm

Let $[1, n]$ denote the range of integers $1, 2, \ldots, n$ and $[0, 1]$ denote the range of all real values between 0 and 1 included. We use a uniform hash function $h : [1, n] \to [0, 1]$. Note that such a function actually requires $\Theta(n)$ words of space to be stored: the algorithm is not practical, but we describe it for its simplicity.

Counting distinct elements of a set

Idealized Flajolet-Martin's algorithm

We simply initialize $y = 1$ and then for each stream element x, update $y = \min(y, h(x_i))$. When the stream ends, return the estimate $d' = 1/y - 1$. Think about the intuition behind that!

Lemma

Let $y = \min\{h(x_1), \ldots, h(x_m)\}$. Then, $E[y] = 1/(d+1)$.

Counting distinct elements of a set

Proof

$$
\begin{aligned}
E[Y] &= \int_0^1 P(y \geq \lambda) d\lambda \\
&= \int_0^1 P(\forall x_i : h(x_i) \geq \lambda) d\lambda \\
&= \int_0^1 (1-\lambda)^d d\lambda \\
&= \left. \frac{(1-\lambda)^{d+1}}{d+1} \right|_0^1 \\
&= \frac{1}{d+1}
\end{aligned}
$$

Counting distinct elements of a set

Unfortunately, in general $E[1/y] \neq 1/E[y]$. On the other hand, if y is very close to $1/(d+1)$, then intuitively also d' will be very close to d. We will prove this intuition by studying the relative error of y with respect to $1/(d+1)$, and then turn this into a relative error of d' with respect to d.

Lemma

Let $y = \min\{h(x_1), \ldots, h(x_m)\}$. Then, $Var[y] \leq \frac{1}{(d+1)^2}$.

Counting distinct elements of a set

Improving the variance

From here, we use the boosted Chebychev. We define an algorithm $FM+$ that computes the mean $y' = \sum_{i=1}^{s} y_i/s$ of s independent parallel instances of FM, for some s to be determined later. Applying Chebyshev, we obtain

$$P(|y' - \frac{1}{d+1}| > \frac{\epsilon}{d+1}) \leq \frac{1}{(d+1)^2} \cdot \frac{(d+1)^2}{s\epsilon^2} = \frac{1}{s\epsilon^2}$$

Taking the median

Our algorithm FM+ returns $d' = 1/y' - 1$. How much does this value differ from the true value d? We can set the parameters (omitted) such that the failure probability is at most $1/3$. Then our final algorithm FM++ runs $t = 72ln(1/\delta)$ parallel instances of FM+ and returns the median result. From the above median Lemma we obtain the following theorem:

Counting distinct elements of a set

Theorem

For any desired relative error $0 < \epsilon \leq 1$ and failure probability $0 < \delta < 1$, with probability at least $1 - \delta$ the FM++ algorithm counts the number d of distinct elements in the stream with relative error at most ϵ,
i.e. it returns a value d' such that:

$$P(|d' - d| > \epsilon \cdot d) \leq \delta$$

In order to achieve this result, during its execution FM++ needs to keep in memory $O(\frac{ln(1/\delta)}{\epsilon^2})$ hash values.

HyperLogLog

The LogLog family

The original algorithm by Flajolet and Martin is based on the following idea: map each element to a q-bits hash $h(x_i)$, remember the maximum number $l = lb(h(x_i))$ of leading 0-bits seen in any $h(x_i)$, and finally, return the estimate 2^l .

Example

$lb(00111010) = 2$. The idea is that, in a set of hash values of cardinality $d = 2^l$, we expect to see one hash $h(x_i)$ prefixed by l zeroes. It is not hard to see that our idealized algorithm presented above is essentially equivalent to this variant.

HyperLogLog

The LogLog family

Durand and Flajolet later refined this algorithm, naming it *LogLog*. The name comes from the fact that one instance of the algorithm needs to store in memory only l, which requires just $\log \log d$ bits. When using k "short bytes" of $\log \log d$ bits (in practice, 5 bits is sufficient), the algorithm computes a $(1 \pm 1.3/k)$ approximation of the result with **high probability**.

Using only one hash function

Instead of using several independent hash functions, Durand and Flajolet use a single hash function but use part of its output to split values into one of many buckets. To break the input entry into m buckets, they suggest using the first few (k) bits of the hash value as an index into a bucket M_i and compute the l on what is left.

HyperLogLog

Example

For example, assume the hash of our incoming datum looks like hash(input)=1011011101101100000. Let's use the four leftmost bits ($k = 4$) to find the bucket index. So that input should update the 11th bucket. From the remaining, 011101101100000, we can obtain the longest run of consecutive 0s from the leftmost bits, which in this case is one. Thus, we would update bucket number 11 with a value of 1.

Using this procedute the cardinality is estimated as $d = c \cdot m \cdot 2^{\frac{1}{m}\sum M_i}$. The arithmetic mean can be assumed to be near $\log d/m$ plus an additive bias.

For m buckets, this reduces the standard error of the estimator to about $1.3/\sqrt{m}$. Thus, with 2048 buckets where each bucket is 5 bits (which can record a maximum of 32 consecutive 0s), we can expect an average error of about 2.8 percent; 5 bits per bucket is enough to estimate cardinalities up to 2^{27} per the original paper and requires only $2048 \cdot 5 = 1.2MB$ of memory. That's pretty good for basically $1KB$ of memory.

HyperLogLog

The HyperLogLog change

In 2007, Flajolet, Fusy, Gandouet, and Meunier further improved the approximation to $(1 \pm 1.04/k)$ by only retaining the 70 percent smallest values and discarding the rest for averaging (this algorithm is called *SuperLogLog*). *HyperLogLog* then uses a harmonic mean of the estimates and can be shown to use $O(\log\log d + \log d)$ space.

Then the cardinality from m buckets can be estimated as $c \cdot m^2 / \sum_{j=1}^{m} 2^{-M_j}$, where $c = 0,794$ is a constant to counterbalance the bias.

Here is the intuition underlying the algorithm. Let d be the unknown cardinality. Each substream will comprise approximately d/m elements. Then, its maximum should be close to $log_2(d/m)$. The harmonic mean $m/\sum_{j=1}^{m} 2^{-M_j}$ of the quantities 2^{M_i} is then likely to be of the order of d/m. Thus, $m^2/\sum_{j=1}^{m} 2^{-M_j}$ should be of the order of d.

HyperLogLog

Set union operations are straightforward to compute in HLL and are lossless. To combine two HLL data structures, simply take the maximum corresponding bucket values of the two and assign that to the corresponding bucket in the unioned HLL. If we think about it, it is exactly as if we had fed in the union of the two data sets to one HLL to begin with. Such a simple set union operation allows us to easily parallelize operations among multiple machines independently using the same hash function and the same number of buckets.