Algorithms and Probability (FS2025)
Week 8

Rui Zhang

April 8, 2025

Contents

1 Exercise Feedback

- The majority got everything right. Props to you guys! I am very happy to see that you have very good intuition when it comes to probability
- To formally derive that the probability of getting a card from a new group, I recommend writing a longer, more formal derivation. For example, a sentence like "After receiving i groups of cards, the probability of receiving a card from a group you already own is $(ki)/(kn) = i/n$ because there are a total of k cards per group. Thus the probability of getting a new group is $1 - i/n$ and we get:"

  $$X_i \sim \text{Geo}(1 - i/n)$$

  "Where X_i is the number of rounds needed to get the $i + 1$st group."

  This assures that in an exam, nobody can nitpick you and deduct points.

2 Content

2.1 Variance

After establishing everything around expectation, which you can intuitively remember as the weighted mean of all possible values of a random variable, we would like to look at some other metrics of random variables. In particular, when analyzing data, we would also like to know how much the data varies from this mean that we have calculated:

Definition: Let X be a random variable with expectation $\mu = \mathbb{E}[X]$. We then define the variance of X to be

$$\text{Var}[X] := \mathbb{E}[(X - \mu)^2]$$

and we call the square root of this value the standard deviation $\sigma = \sqrt{\text{Var}[X]}$

After a bit of calculation, you can also get:

Theorem: $\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

The second formula has more applications that one would assume at first. We will look at an exercise in class, if there is time. We also have

Theorem: $\text{Var}[a \cdot X + b] = a^2 \cdot \text{Var}[X]$

The fact that b disappears can be visualized in your head: Imagine adding a constant value b to a random variable X. Since all values are shifted by a certain constant amount, it does not change the overall amount in which the values vary from the expectation.

Theorem: Let $X_1, \ldots X_n$ be n mutually independent random variables and X equal to the sum of all of those random variables. Then we have that:

$$\text{Var}[X] = \text{Var}[X_1] + \cdots + \text{Var}[X_n]$$

Note that no matter the nature of your indicator variables, we can never guarantee that the variance of products of random variables is equal to the product of variance of those random variables.

Exercise: (Doublesums)

We have blue and red beads in a pot and want to build a necklace with n beads. To do this, we randomly draw a bead from our pot and add it to the necklace. We assume that we have an infinite supply of beads and that each draw results in a red bead with probability 0.25 and a blue bead with probability 0.75. After adding n beads in sequence to our necklace (where the beads are numbered in order), we close it into a loop. We now ask how many color transitions occur in the necklace, meaning how many positions exist where the i-th bead has a different color than the $i+1$-th bead (where the $n+1$-th bead is the same as the 1st bead, since it forms a loop). Let

$$X := \text{Number of color transitions in the necklace}$$

To start, we first define indicator variables for all subproblems (which is almost always a good approach) to simplify the problem. One can verify (or look it up in a reference) that X does not follow a binomial distribution.

$$X_i := \text{Indicatorvariable for the event "There is a color transition at the i-th bead"}$$

(a) What is $\mathbb{E}[X]$?

(b) What is $\mathbb{E}[X_i \cdot X_j]$ for arbitrary $1 \leq i \leq j \leq n$? (Hint: Case Distinction)

(c) Using (b), what is $\text{Var}[X]$?

2.2 Inequalities

We have done a lot of math now. However, this is not a pure maths course. We want to be able to make statements about algorithms using probabilities. To this end, we would like to do something like define the runtime of an algorithm as a random variable and be able to say that the probability of this random variable being greater than some number of iterations is less than some very very small number. (i.e. The probability of our algorithm being trash is very unlikely.)

> **Theorem:** (Markov) Let X be a random variable **which only attains non-negative values**. Then we have for all $t \in \mathbb{R}^{>0}$
> $$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$$

To quote Professor Steger, who has sadly left ETH: Proof is via leaving stuff out.

> **Theorem:** (Chebyshev) Let X be a random variable and $t > 0$. Then we have:
> $$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{Var}[X]}{t^2}$$

which formalizes the intuition that the probability that your random variable is ”far away” from the expectation is small and depends on the variance.

These two inequalities are pretty good. However, if you have a binomial distribution or n independent bernoulli random variables, then we can get even better bounds. Whenever in an exercise you cannot get the bound you want, check if your random variable is a binomial random variable. Because then you can apply Chernoff:

> **Theorem:** (Chernoff). Let $X_1, \ldots, X_n$ be n independent Bernoulli random variables such that $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. Then for the sum of those independent random variables $X = X_1 + \cdots + X_n$, we have:
> $$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2\mathbb{E}[X]} \text{ for all } 0 < \delta < 1 \tag{1}$$
> $$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2\mathbb{E}[X]} \text{ for all } 0 < \delta < 1 \tag{2}$$
> $$\Pr[X \geq t] \leq 2^{-t} \text{ for } t \geq 2e\mathbb{E}[X] \tag{3}$$

Exercise: (Continuation of Doublesums)

(d) Find a best possible upper bound for $Pr[X \geq t + \mathbb{E}[X]]$ via Markov.

(e) Find a best possible upper bound for $Pr[X \geq t + \mathbb{E}[X]]$ via Chebyshev.

2.3 Gambling (on Algorithms)

Figure 1: 99% of gamblers stop before they win big. Never give up.

A standard algorithm consists of an input fed into the algorithm, which then gives you an output. For all inputs I of length n we have that the runtime is in some function of n: $O(f(n))$ Introducing: random algorithms. They take not only an input I from the user, but also has access to random bits / random numbers. The output $\mathbb{A}(I, R)$ is both dependent on I and the random numbers R. We assume that the random bits / random numbers generated by a random number generator are mutually independent and would like to prove statements of the form:

$$\text{Correctness: For all inputs } I\text{, we have } \Pr[\mathbb{A}(I, R) \text{ is correct}] \geq \text{something that is almost 1} \tag{4}$$
$$\text{Runtime: For all inputs } I\text{, we have } \Pr[\text{Algo. Runtime} \leq O(f(n))] \geq \text{something that is almost 1} \tag{5}$$

Random Algorithms are divided up into two types:

Monte-Carlo-Algorithm:	Las-Vegas-Algorithms
Correctness of the result is a random variable.	Runtime is a random variable.

And our goal for each type of algorithm is to reduce the probability of the corresponding random variable being something undesirable by simply repeating the algorithm multiple times.

2.3.1 Las-Vegas Algorithms

Las-Vegas Algorithms can be seen in two ways, which makes it kind of confusing sometimes. Either see it as an algorithm which runs for a random amount of time. Or alternatively, see it as an algorithm that can sometimes return "???" because we wrote code that stopped this algorihtm before it could finish. These two definitions are equivalent: Take an algorithm that sometimes returns "???" and repeat it until it repeats something sensical within its alloted time, then we get the standard definition again.

Theorem: Let A be an algorithm which never gives a wrong result but sometimes returns "???", such that

$$\Pr[A(I) \text{ correct}] \geq \epsilon$$

Then for all $\delta \geq 0$ let A_δ be the algorithm which repeats A until a non-"???" value is returned or until at most $N = \epsilon^{-1} \ln \delta^{-1}$ repetitions have occurred (in which case A_δ gives up and returns "???" too.) Then, A_δ has the correctness probability:

$$\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$$

Proof. The probability that "???" is returned N times (and A_δ thus also returns some nonsense), is

$$\begin{aligned} \Pr[A_\delta(I) incorrect] &= (1-\epsilon)^N \\ &\leq e^{-\epsilon\epsilon^{-1} \ln \delta^{-1}} \\ &= e^{\ln \delta} \\ &= \delta \end{aligned}$$

□

2.3.2 Monte-Carlo Algorithms

For Monte-Carlo-Algorithms, we cannot always make the same improvements. The main problem here is that we cannot know if the returned value from the algorithm is correct or not. However, under certain assumptions, we can improve via the following two theorems:

Theorem: Let A be a randomized algorithm which gives a binary output: Either "Yes" or "No", where we have a "one-sided" error, i.e.:

$$\Pr[A(I) = "Yes"] = 1 \text{ if } I \text{ is a "Yes-Instance" (so the algorithm should return "Yes")} \quad (6)$$
$$\Pr[A(I) = "No"] \geq \epsilon \text{ if } I \text{ is a "No-Instance" (so the algorithm should return "No")} \quad (7)$$

Then for $\delta > 0$ let A_δ be an algorithm which repeats A until either "No" is returned (in which case A_δ returns "No") or until $N = \epsilon^{-1} \ln \delta^{-1}$ iterations have passed, which all resulted in a "Yes". Then we have for all inputs I:

$$\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$$

Proof. Left as an exercise. It is basically the same as with Las-Vegas-Algorithms □

What about if we have errors on both sides?

Theorem: Let $\epsilon > 0$ and A be a randomized algorithm which either returns "Yes" or "No". Where, independent of the instance, we have:

$$\Pr[A(I) \text{ correct}] \geq 1/2 + \epsilon$$

Then we have for all $\delta > 0$ Let A_δ be the algorithm that repeats $N = 4\epsilon^{-2} \ln \delta^{-1}$ iterations of A and outputs the majority of answers, then we have that

$$\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$$

Think about it like this: if the probability of the original algorithm being correct is larger than 1/2, then out of the N iterations we will get more correct responses than wrong responses. If the original probability is equal to 1/2, then on average we actually have an equal number of correct responses and wrong responses. In this case, the algorithm would pick a random output. If we had a probability of less than 1/2, then we would theoretically have less correct answers than wrong answers.

2.3.3 Optimization Algorithms

Finally, I would like to visit the case where we would like to maximize the result of a random algorithm. In this case, the returned value of an algorithm should be larger than some baseline $f(I)$. We may now repeat this algorithm until we find a value bigger than $f(I)$ just like with the ideas from before:

Theorem: Let $\epsilon > 0$ and A be a randomized algorithm for an optimization problem, where we have

$$\Pr[A(I) \geq f(I)] \geq \epsilon$$

Then for all $\delta > 0$ let A_δ be the algorithm which repeats A $N = \epsilon^{-1} \ln \delta^{-1}$ times and returns the best result of all iterations. Then we have for A_δ that

$$\Pr[A_\delta(I) \geq f(I)] \geq 1 - \delta$$