Assignment 2: CS 215, Fall 2024

Prof. Sunita Sarawagi

Released: 31 August, 2024
Hints and Soft Deadline: 9 September, 2024
Hard Deadline: 11:59 PM, 13 September, 2024

Please read all instructions carefully!

Instructions

1. **Report instructions**. You are requested to type out your solutions in LaTeX. If this is not possible, you may print out `a2_hand.pdf`, hand-write all solutions neatly in the space provided and submit a pdf containing scanned copies of the filled-in document. Again, please note that humans aren't perfect image-to-text converters; do write large and clear!

2. **Report instructions, continued**. The submitted file must contain the names and roll numbers of all group members on the first page. Some problems will require code; please make sure adequate running instructions are included with your solution to each such problem.

3. **Code file instructions**. For each code file: type out the name and roll number of each group member in a comment at the top of the file, along with the problem number the code is for.

4. **And then, check format**. Once the pdf and code are ready, `zip` them into one folder with the name `N=A2-RollNumberOfFirstStudent-RollNumberOfSecondStudent-RollNumberofThirdStudent`. Please name the zip file `N.zip`. You may use the command `zip -r N.zip N` from the directory containing your submission directory `N` to do this. If the assignment was done in a group of size not equal to 3, the name of the folder and zip file will be `A2` followed by the roll numbers of each member; each item separated by hyphens. Any letters in the roll number must be capitalized. The submissions for this assignment **MUST** follow the given directory structure and assignments not adhering to this risk not being graded.

```
N
├── report.pdf
├── code
│   ├── 2c.py
│   ├── 2d.py
│   └── 3.py/ipynb
└── images
    ├── 2c.png
    ├── 2d1.png
    ├── 2d2.png
    ├── 2d3.png
    ├── 3b.png
    ├── 3c.png
    ├── 3d.png
    └── 3f.png
```

5. **- and submit (on time!)**. Upload the file on moodle BEFORE 11:59 pm on the due date. No assignments will be accepted thereafter. Please expect Murphy's Law to hold post 11.45 pm on the due date. Note that **only one** student per group should upload their work on moodle, though all group members will receive grades.

6. **Misc 1: Bonuses**. Some tasks in some questions are optional and are marked with a **B**. Solving some of these questions can get you extra points, capped to the maximum number of points on the assignment.

7. **Misc 2: Hints**. After a little over a week, hints will be posted to some tasks of some questions (precisely the tasks marked with a ⋆). All submissions that are not updated on moodle at any point after the soft deadline will receive 10 credits of bonus points (capped off at the maximum number of points for the assignment), for not using the hints. Use the trade-off of scoring 10 points without hints and perhaps scoring more than that with a few well-placed hints wisely!

8. **Total**. The maximum possible marks on this assignment are 100 with the marks being normally distributed between questions, that is, $Q1$ and $Q5$ carry 15 marks each, $Q2$ and $Q4$ carry 20 marks each and $Q3$ carries 30 marks. Let us know what you think the "most probable" mean and variance is for this Gaussian "histogram" ;-P

9. **Plagiarism**. This assignment is mostly for your own learning, and you are free to discuss the problems with anyone. However, verbatim copying of solutions is unacceptable: compromising your credibility for $5-10$ points in one course (whose grade will not even matter in a few years from now) is not worth it by a mile. Please give the problems your time; they are meant to help you understand the subject better.

10. **One last thing**. Please preserve a copy of all your work until the end of the semester. And please have fun!

§ 1 Mathemagic

We explore some connections between some special random variables, via the notion of probability generating functions.

Definition 1 (PGF, MGF). *Let X be a random variable taking on only non-negative-integer values. Suppose that X is distributed according to probability mass function P. We define the probability-generating-function of the distribution $P[X]$ of random variable X by*

$$G(z) := \mathbb{E}\left[z^X\right] = \sum_{n=0}^{\infty} P[X = n]\, z^n.$$

The moment-generating-function *of the same distribution is defined by $M(t) := G(e^t)$ for t such that the series for G converges.*

◊ Task A [1]

Derive the PGF when X is a Bernoulli random variable with parameter p, that is, $X \sim \text{Ber}(p)$. Call this PGF G_{Ber}.

◊ Task B [2]

Let G_{Bin} the PGF when $X \sim \text{Bin}(n, p)$. Show that $G_{\text{Bin}}(z) = G_{\text{Ber}}(z)^n$.

◊ Task C (⋆) [4]

Suppose that $X_1, X_2, \cdots, X_k$ are independent non-negative-integer-valued random variables, each distributed with the same probability mass function P. Let G be their common PGF. Consider random variable $X = X_1 + X_2 + \cdots + X_k$ defined on the cartesian product of the sample spaces underlying the random variables X_i. Let the PGF corresponding to X be G_Σ. Show that $G_\Sigma(z) = G(z)^k$.

◊ Task D [1]

Consider now the Geometric distribution. Let $X \sim \text{Geo}(p)$. Derive its PGF.

◊ Task E [3]

Consider $X \sim \text{Bin}(n, p)$ and $Y \sim \text{NegBin}(n, p)$. Let their PGFs be $G_X^{(n,p)}(z)$ and $G_Y^{(n,p)}$ respectively. Show using previous tasks or otherwise that for every $0 < p < 1$, we have

$$G_Y^{(n,p)}(z) = \left(G_X^{(n,p^{-1})}(z^{-1})\right)^{-1}.$$

That shows that the negative binomial distribution is not only morally an "inverse" of the binomial distribution (in that it models number of trials while fixing number of successes, while the binomial fixes the number of trials and models the number of successes), but negates the binomial distribution in every way possible! There is more, see Task F below.

◊ Task F [2]

We shall derive the negative binomial theorem using the result from Task E. First, we generalize the binomial coefficient: let $\alpha \in \mathbb{R}$ and $k \in \mathbb{N}$. Then

$$\binom{\alpha}{k} := \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}.$$

Theorem 2 (Binomial theorem, negative exponent). *Let $n \in \mathbb{N}$ and $|x| < 1$. Then*

$$(1+x)^{-n} = \sum_{r=0}^{\infty} (-1)^r \binom{n+r-1}{r} x^r = \sum_{r=0}^{\infty} \binom{-n}{r} x^r.$$

It is almost (the summation not being finite is the only difference) as if one could simply put a negative number for the exponent in the usual binomial theorem.

◊ Task G [2]

An easy consequence of all the hard work. Suppose that the PGF of random variable X is $G(z)$. Show that the expectation of X is simply the derivative of G at 1, i.e. $\mathbb{E}[X] = G'(1)$. Using this and the PGFs constructed previously, derive the means of the Bernoulli, Binomial, Geometric and Negative binomial distributions as a function of their parameters.

§ 2 Normal Sampling

It's nice and all to know that standard normal random variables exist and are well-behaved; but can we actually sample from a standard normal distribution?

Let's be clear on what we want to do. We would like to find an algorithm $\mathcal{A}$ that takes some uniformly random numbers in $[0, 1]$, and generates an output $x \in \mathbb{R}$. The algorithm should use the randomness of the numbers it receives in such a way that the output x of the algorithm is distributed standard normally. Denote by $f_{\mathcal{A}}(x)$ the PDF of the algorithm's output (a random variable). We wish that for every $x \in \mathbb{R}$,

$$f_{\mathcal{A}}(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

To find such an algorithm, we will show and then use a very general theorem. A note on notation: we shall use f_X to denote the PDF of random variable X, and F_X to denote the CDF.

◊ Task A [2]

Theorem 3. *Let X be a continuous real-valued random variable with CDF $F_X : \mathbb{R} \to [0, 1]$. Assume that F_X is invertible. Then the random variable $Y := F_X(X) \in [0, 1]$ is uniformly distributed in $[0, 1]$.*

Prove the theorem above.

◊ Task B [2]

Suppose now that we are given random variable Y which is uniform over $[0, 1]$. Explain how an algorithm $\mathcal{A}$ may be constructed taking as input a sample y according to the distribution of Y (so the algorithm is given just one uniformly random number in $[0, 1]$), such that for every $u \in \mathbb{R}$, we have

$$F_{\mathcal{A}}(u) = F_X(u).$$

In other words, the output of $\mathcal{A}$ has the same cumulative distribution function as X, and upon taking the derivative so has the same PDF.

◊ Task C [8]

In this task, you will use the `numpy.random` module to generate random numbers uniformly in $[0, 1]$ and the `norm` class (gives you access to F_X and F_X^{-1} for a Gaussian random variable X) from `scipy.stats` to sample from a Gaussian. In particular,

- Write a Python function `sample(loc, scale)` that samples from the Gaussian with mean at $x =$ `loc` and standard deviation `scale`.
- Generate $N = 10^5$ independent samples using the function above for the four parameter choices $(\mu, \sigma^2) = (0, 0.2), (0, 1.0), (0, 5.0), (-2, 0.5)$.
- Plot the samples for each parameter choice using `matplotlib.pyplot`. You should roughly reproduce the shape of the plot of the four Gaussians from the lecture slides, see figure 1). The bottom plot is a plot of the samples; it mimics the top plot of the different PDFs, confirming that we have indeed sampled from the different Gaussian PDFs. Save the plot in `2c.png`.

You may not use the built in `numpy.random.normal` or `random.randn` functions that directly sample from Gaussians for this task. For the plot of the samples, you may not use kernel-density estimation (`kde`) tools; `matplotlib.pyplot` by itself is sufficient. Please write all your code for this task in one file called `2c.py`.

◊ Task D (⋆) [8]

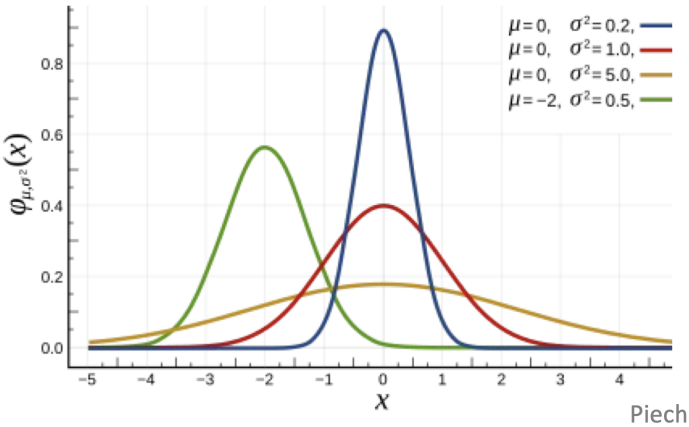
Now we consider another way to sample normal distributions - approximately - using a Galton board (see figure 2 for a picture). Imagine a ball that starts at the top of a Galton board. As it hits the first peg, it moves to the left of the peg with probability $1/2$ and to the right of the peg with probability $1/2$. After falling vertically downwards a little, it hits another peg. Again, it moves left or right of the new peg with equal probability. Suppose it makes h collisions with pegs before reaching the bottom of the Galton board. We call h the depth of the board. At the end, the ball falls into one of the wood-piece-separated pockets. There are $h + 1$ pockets at the bottom of the board, and each ball falls into exactly one of these as per the random directions it chose at each collision.

Consider simulating the motion of a large number N of balls along the Galton board, and record the fraction of balls that finally end up in each of the $h + 1$ pockets.

The simulation works by simulating the motion of each ball from the top to bottom of the Galton board. Initially the ball is at $x = 0$, and in the first step the ball moves left or right with equal probability - so its current position x is incremented or decremented with equal probability. The second step is identical - with a different starting position x. Suppose in the first step x was incremented, so $x = 1$ now. Then on the second step, x is decremented (back to 0) or incremented (to 2) with equal probability. Perhaps it was incremented again. The third step decrements it to 1 or increments it to 3 with equal probability. Perhaps it was decremented to 1. A fourth step takes it to 0 or 2 with equal probability. And so on. h steps ensue till the final value of its position x is the pocket the ball will fall into.

The choice of moving left or right can be simulated by simulating from $\{0, 1\}$ uniformly randomly (find a function in `numpy.random` to do this). h uniform samples from $\{0, 1\}$ thus allow us to simulate one ball.

The process can be repeated N times with the final pockets of each simulated ball being recorded, to yield a simulation of the Galton board. Note that it is possible, using `numpy`, to omit any Python loops. You are not required to omit loops in your code but it needs to be fast enough to gather the data required.



Piech & Cain, CS109, Stanford University

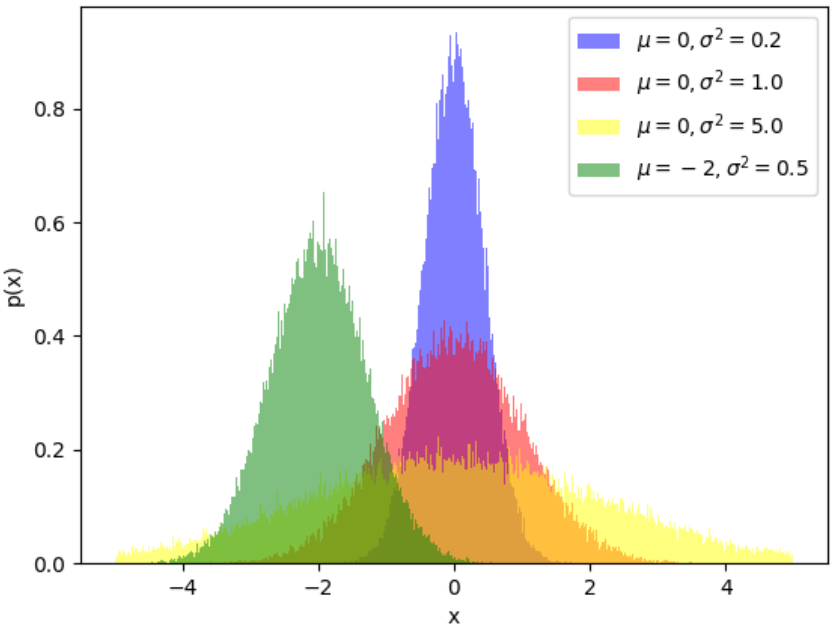


Figure 1: Question 2, Task C: The Bell curve, from uniformly random numbers

Your task is to carry out this simulation for $N = 10^5$ (if this is too large, you may use $N = 10^3$) for the three values $h = 10, 50, 100$ of the depth of the board. For each value of h, plot the counts of balls in each pocket obtained as a histogram, with one bin for each pocket. The code is to be written in file `2d.py`, with the three histograms saved to files `2d1.png`, `2d2.png` and `2d3.png` (see figure 2 for a reference plot). What do you notice about the shape of the tops of the histogram?

◊ Task E (B) [5]

The goal here is to theoretically show that the distribution of number of balls in each pocket is normally distributed. Suppose that $h = 2k$ is even. Consider a Galton board of depth h. Let random variable $X \in \{-h, -h+2, \cdots, 0, 2, \cdots, h-2, h\}$ describe the pocket in which a ball finally lands (notice that X can only be even). The probability mass distribution $\mathrm{P}_h[\cdot]$ of X is determined by the randomness of the outcome of each collision. There are two sub-tasks here.

– Compute (in closed form) the value $\mathrm{P}_h[X = 2i]$ for each $i \in \{-k, -k+1, \cdots, k-1, k\}$.

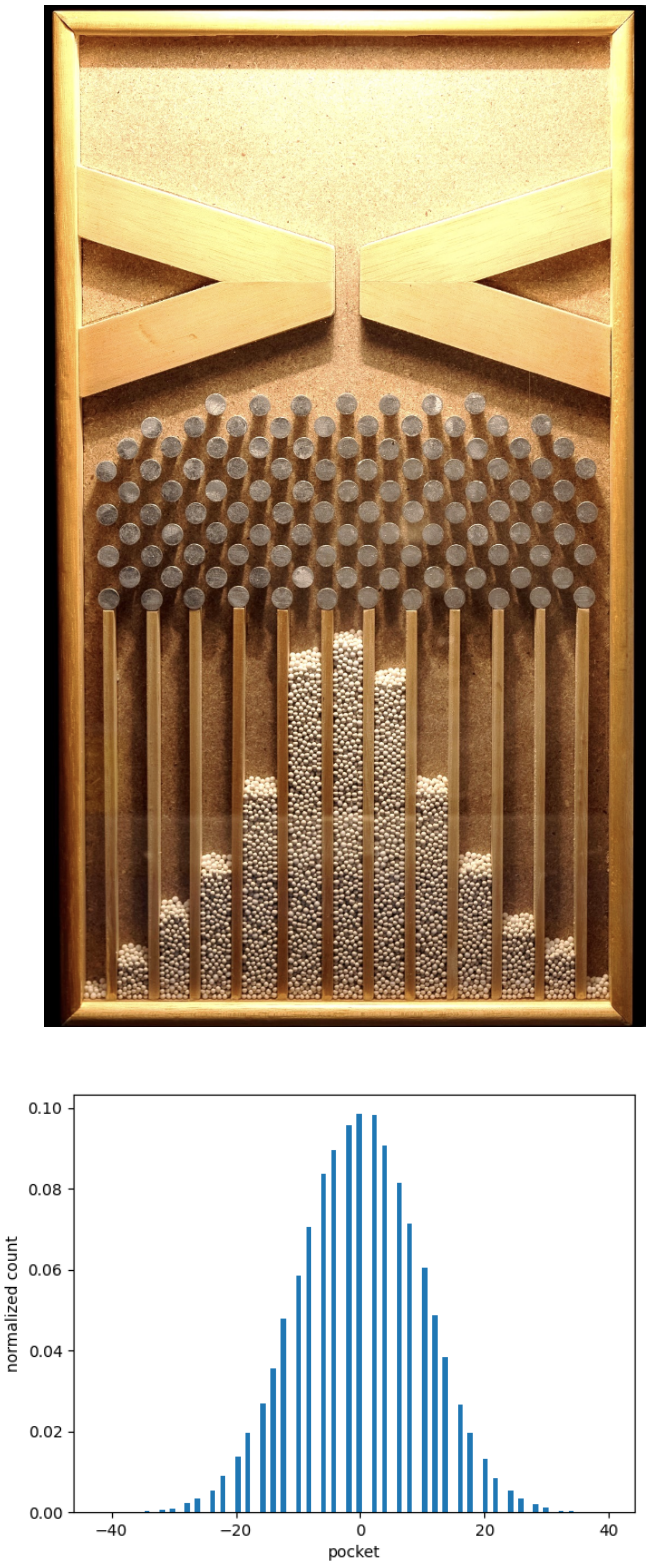


Figure 2: (top) A Galton board with $h = 10$ (ignore the leftmost and rightmost pockets). (bottom) A simulation with $N = 10^5$ and $h = 100$.

– Show that for large enough h and even $i \ll \sqrt{h}$,

$$\mathrm{P}_h[X = i] \approx \frac{1}{\sqrt{\pi h}} e^{-\frac{i^2}{h}} = \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{h}{2}\right)(i).$$

You may need to use Stirling's approximation for the factorial

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

to solve this subtask.

§ 3 Fitting Data

In this problem, we are given a dataset that came from some distribution that we don't know offhand, and we want to find what that distribution is. Why? So that we can then predict future outputs that are sampled from the same distribution, if we get close enough to the right answer.

Suppose we have a dataset consisting of n samples $S = \{x_1, \cdots, x_n\}$, with each $x_i \in \mathbb{R}^d$. You are given a dataset which we will call $\mathcal{D}$ in the file `3.data`. We assume that there is an underlying probability distribution function P for a random variable X such that each x_i was sampled independently from $\mathrm{P}[X]$: the i^{th} sample is a random variable X_i with the same distribution as X, i.e. $\mathrm{P}[X_i = x_i] = \mathrm{P}[X = x_i]$. Notice that a dataset may then simply be treated as the collection of outcomes of a number n of experiments, each of which consist of sampling X from its underlying distribution P.

Guessing or estimating the distribution P will tell us roughly what $X_{n+1}, X_{n+2}, \cdots$ might turn out to be. This is useful, and so trying to estimate an unknown distribution from a sample will be our goal for this question.

Code for all subparts is to be written in `3.py`. If you wish, you may choose to display plots with the code, using an `.ipynb` file. Please name it `3.ipynb` in this case.

◊ Task A [2]

The ith moment of a random variable is defined by $\mu_i := \mathbb{E}\left[X^i\right]$. When given a sample of n datapoints, we define the sample ith moment as the moment of the random variable that is each datapoint with equal probability $1/n$. In particular, the sample ith moment is

$$\hat{\mu}_i := \frac{x_1^i + \cdots + x_n^i}{n}.$$

Compute the first two ($i = 1, 2$) moments of $\mathcal{D}$. You may use `numpy` arrays and operations on them, but no loops are allowed to compute the moments (yes, it can be done with `numpy`).

◊ Task B [2]

Let's first try to graphically guess the distribution from our dataset.

Recall that a dataset is simply the collection of outcomes of many independent identical experiments, and that probability of an element x models the fraction of experiments with outcome x.

Compute a histogram of the dataset and plot it. From the histogram, graphically guess the normal distribution parameter μ. Please attach any code used in file `3.py`, saving the histogram computed in file `3b.png`.

A histogram is typically used to "see" what kind of distribution the sample could have come from.

◊ Task C (⋆) [1+2+2]

We have still got nowhere close to finding a distribution that fits the data we have well. Let's guess. It looks reasonably like a binomial distribution centred near about 6. Let us find the binomial distribution that is closest to our data.

One way to define closeness is to ask for the first few moments to be equal (it is a theorem in statistics that if every moment of two distributions are identical, then the two distributions must be identical as well - we are roughly approximating this theorem here).

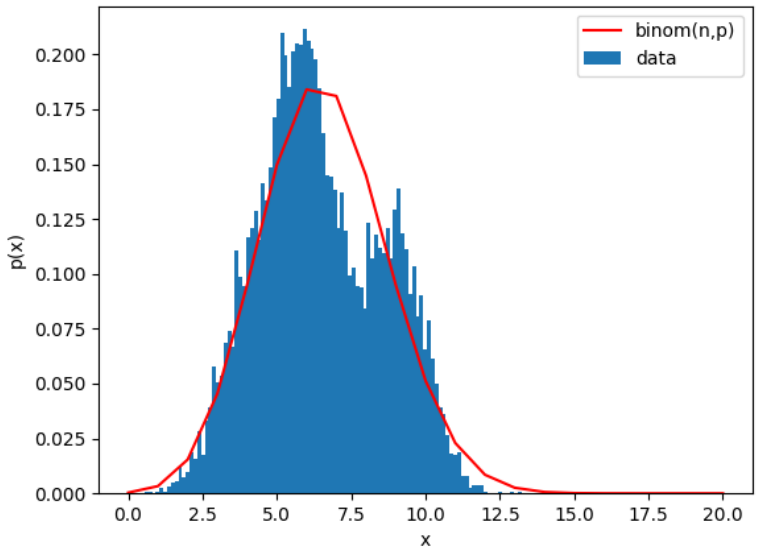


Figure 3: The best binomial distribution approximation to the true distribution

In this task, we will find the best-fit binomial distribution to $\mathcal{D}$ by asking for the first two moments $\hat{\mu}_1$ and $\hat{\mu}_2$ to be equal to the corresponding two moments of $\text{Bin}(n, p)$, for some suitable choice of n and p.

1. First, compute an expression for the first two moments $\mu_1^{\text{Bin}}, \mu_2^{\text{Bin}}$ of the distribution $\text{Bin}(n, p)$ as a function of n and p.

2. Then, use the `fsolve` function from `scipy.optimize` to compute a solution (n, p) to $\hat{\mu}_i = \mu_i^{\text{Bin}}$, $i = 1, 2$. Round n to either $\lfloor n \rfloor$ or $\lceil n \rceil$ based on which one satisfies the equalities better (they will not be exactly satisfied). Say the found parameters are (n^*, p^*).

3. Finally, using `numpy.linspace` and `scipy.stats.binom.pmf`, plot the binomial distribution $\text{Bin}(n^*, p^*)$ on top of the histogram of $\mathcal{D}$. You should get something like the figure in figure 3. Save the plot to `3c.png`.

◊ Task D [3+2+2]

Well, that was not too bad an approximation. Let us try another distribution, this time continuous.

The gamma distribution is a two-parameter family of continuous probability distributions. It is parameterized by two parameters: shape parameter k and scale parameter θ. Its probability density function is given by

$$f(x; k, \theta) = \frac{1}{\theta^k \Gamma(k)} x^{k-1} e^{-\frac{x}{\theta}}$$

where $\Gamma(k) = \int_0^\infty t^{k-1} e^{-t} dt$ is the Gamma function. We now wish to find the best Gamma-distribution approximation to the true distribution of $\mathcal{D}$.

Your task and approach is essentially the same as in Task C. Restated for clarity, you must do the following:

1. First, compute an expression for the first two moments $\mu_1^{\text{Gamma}}, \mu_2^{\text{Gamma}}$ of the distribution $\text{Gamma}(k, \theta)$ as a function of k and θ.

2. Then, use the `fsolve` function from `scipy.optimize` to compute a solution (k, θ) to $\hat{\mu}_i = \mu_i^{\text{Gamma}}$, $i = 1, 2$. No rounding is required, since k, θ may be real in this case. Say the found parameters are (k^*, θ^*).

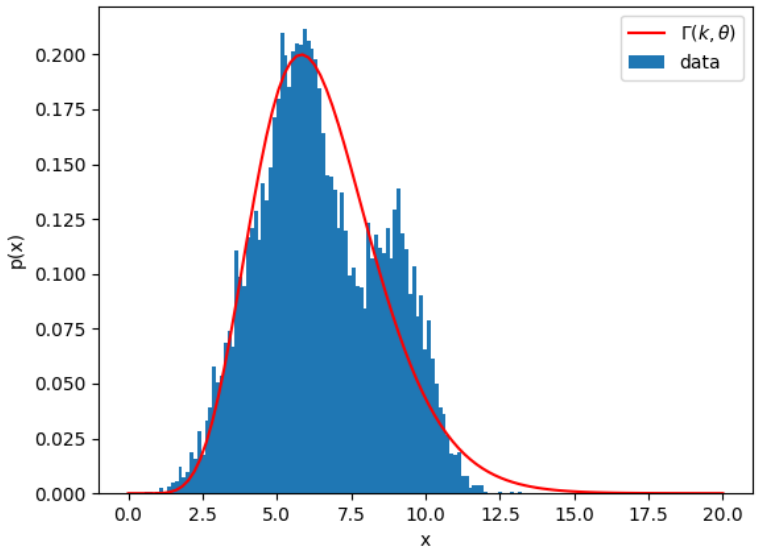


Figure 4: The best Gamma distribution approximation to the true distribution

3. Finally, using `numpy.linspace` and `scipy.stats.gamma.pdf` (the pdf takes three parameters: find out which ones are k and θ and which one is 0), plot the binomial distribution $\mathrm{Bin}(k^*, \theta^*)$ on top of the histogram of $\mathcal{D}$. You should get something like the figure in figure 4. Save the plot to `3d.png`.

◊ Task E (⋆) [3+3]

It looks like both of the distributions did a good job, but which one did a better job?

One simple way to find out is to compute what is called the likelihood of the dataset. This is simply the probability that the dataset would be $\mathcal{D}$, supposing that the true distribution was actually our best-fit distribution.

Definition 4 (Likelihood). *Given a dataset S and a choice of parameter $\lambda = \lambda_0$ for a family of distributions $P[\lambda] \cdot$ parameterized by λ, define the likelihood* of λ_0 *by*

$$\mathcal{L}(\lambda_0 \mid S) := P_{\lambda_0}[S] = \prod_{i=1}^{n} P_{\lambda_0}[X_i].$$

Here $\mathrm{P}_{\lambda_0}[x]$ is the PDF of the distribution whose parameter is λ_0. In our case, $\mathrm{P}_{\lambda}[x]$ is $\mathrm{Bin}(\lambda = (n, p))(x)$ or $\mathrm{Gamma}(\lambda = (k, \theta))(x)$.

For large datasets, since each probability is a small number, the likelihood can underflow. Thus, the average log-likelihood is typically calculated:

$$\ell(\theta \mid S) := \frac{\log \mathcal{L}(\theta \mid S)}{n},$$

where n is the size of the dataset. Calculate (in code, no for loops allowed) the average log-likelihood for both best-fit distributions. Since $\mathrm{Bin}(n, p)(x)$ is nonzero only at integer values of x, you will need to round each datapoint to the nearest integer before computing the likelihood. No such thing required for the Gamma distribution.

A larger likelihood is typically attributed to a better fit. Which distribution was a better fit?

◊ Task F [2+2+2+2]

Notice the two peaks in the distribution? This immediately sort of tells us that the distribution could not have been from a Binomial or Gamma function, since those have a unique mode.

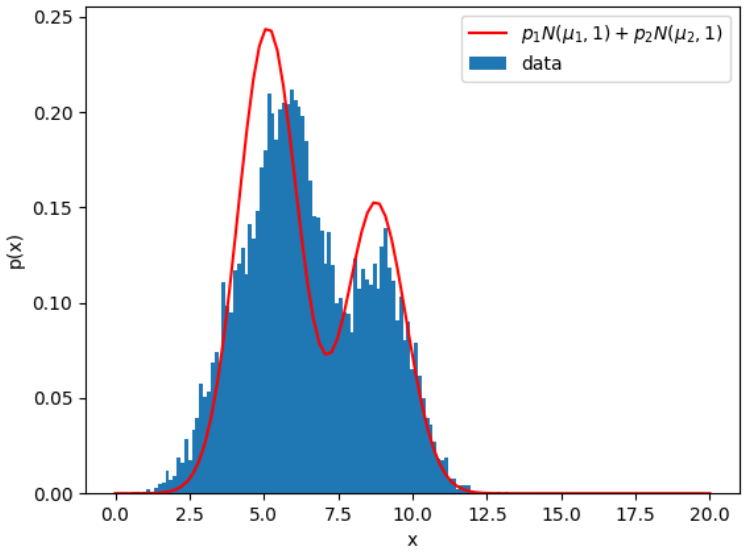


Figure 5: The best two-component unit-variance GMM approximation to the true distribution

A common distribution with two peaks is the Gaussian Mixture Model, which is the subject of Question 5. Please read Task A of Question 5 before moving ahead.

We are going to assume now that our distribution is composed of a two-component Gaussian mixture, each component having variance 1. That is, the distribution modelling our data is assumed to have the pdf

$$\mathrm{P}[x] = \frac{1}{\sqrt{2\pi}}\left(p_1 \exp\left(-\frac{(x-\mu_1)^2}{2}\right) + p_2 \exp\left(-\frac{(x-\mu_2)^2}{2}\right)\right).$$

We have four parameters to find, and so need four moments. To make things easy, here are the first four moments of this distribution (here $\sigma_1 = \sigma_2 = 1$):

$$\begin{aligned}
\mu_1^{\text{gmm}} &= p_1\mu_1 + p_2\mu_2.\\
\mu_2^{\text{gmm}} &= p_1(\sigma_1^2 + \mu_1^2) + p_2(\sigma_2^2 + \mu_2^2).\\
\mu_3^{\text{gmm}} &= p_1(\mu_1^3 + 3\mu_1\sigma_1^2) + p_2(\mu_2^3 + 3\mu_2\sigma_2^2).\\
\mu_4^{\text{gmm}} &= p_1(\mu_1^4 + 6\mu_1^2\sigma_1^2 + 3\sigma_1^4) + p_2(\mu_2^4 + 6\mu_2^2\sigma_2^2 + 3\sigma_2^4).
\end{aligned}$$

As in Tasks C and D, compute the following:

1. First, compute $\hat{\mu}_i$ for $i = 3, 4$.
2. Then, use the `fsolve` function from `scipy.optimize` to compute a solution (μ_1, p_1, μ_2, p_2) to $\hat{\mu}_i = \mu_i^{\text{gmm}}$, $i = 1, 2, 3, 4$. No rounding is required. Say the found parameters are $(\mu_1^*, p_1^*, \mu_2^*, p_2^*)$.
3. Finally, using `numpy.linspace` and `scipy.stats.norm.pdf`, plot the GMM distribution obtained on top of the histogram of $\mathcal{D}$. You should get something like the figure in figure 5. Save the plot to `3f.png`.

To finish, compute the average negative log-likelihood of the obtained GMM distribution. Is it better an approximation than the previous two?

This task shows the power of a more versatile distribution like the GMM; indeed, clustering is typically done this way, using a GMM, the only difference being that an equation solver like `fsolve` is replaced by a heuristic called *expectation maximization*, since solvers are slow for many-component GMMs.

Teaser. A different choice of distribution, with six parameters gave the fit in figure 6. In general, an important part of machine learning is parameter estimation to fit data. Feedforward neural networks typically do just that. As we have seen, more parameters can get you closer to the "right distribution", and it is not surprising that GPT has a few billion parameters that it learns. It is not just roses, though. A large number of parameters runs the risk of overfitting. Data Analysis and ML study better ways to learn parameter values, better families of distributions, how to detect and avoid overfitting, and more.

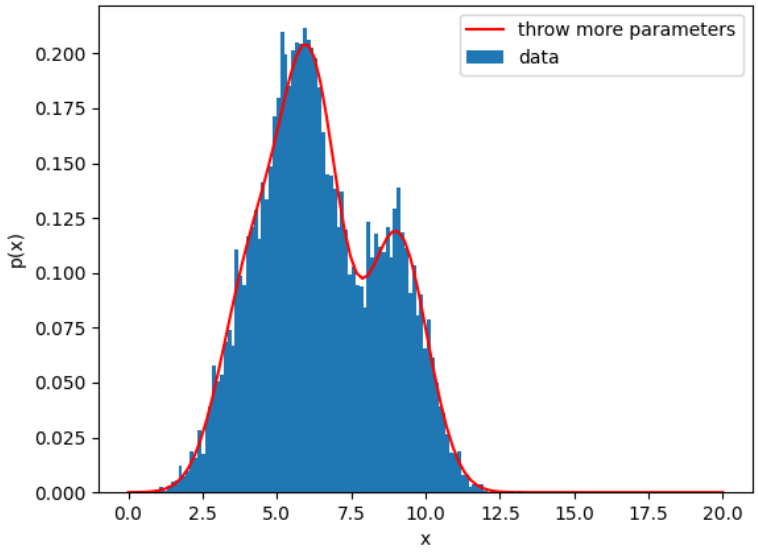


Figure 6: Two more parameters bring the total to 6 parameters estimated using the first six moments

Ungraded Bonus. Any guesses what the true distribution is? Winners can claim a coffee treat at Cafe92 from the authors.

§ 4 Quality in Inequalities

Let us dive deeper into the inequalities we have studied in class (and a new one):

Definition 5 (Markov's Inequality). *Let X be any non-negative random variable and $a > 0$,*

$$P[X \geq a] \leq \frac{\mathbb{E}[X]}{a}.$$

◊ Task A [1+2]

Give an intuitive "proof" for this inequality and reason why it could be correct (you can try playing around with different X's and a's).

Now, prove this inequality rigorously for continuous random variables. Try to reason about the definition of expectation and how you can manipulate it to serve your purpose.

◊ Task B [4]

Now that we have established this inequality, let us move on to linking it to what we have already studied in class - the Chebyshev-Cantelli inequality.

Use Markov's inequality to prove the following version of the Chebyshev-Cantelli inequality for a random variable X with mean μ and variance σ^2: for every $\tau > 0$, we have

$$\mathrm{P}\left[|X - \mu| \geq \tau\right] \leq \frac{\sigma^2}{\sigma^2 + \tau^2}.$$

◊ Task C [3]

Yay, our inequalities are successfully linked! Now we can move onto proving a strong bound through these inequalities...start out by showing that for a random variable X where $M_X(t)$ represents the MGF (see Question 1) for X, the following hold:

$$\mathrm{P}[X \geq x] \leq e^{-tx} M_X(t) \;\; \forall t > 0.$$

$$\mathrm{P}[X \leq x] \leq e^{-tx} M_X(t) \;\; \forall t < 0.$$

◊ Task D (⋆) [1+4+1]

Now take n **independent** Bernoulli random variables $X_1, X_2, \cdots X_n$ where $\mathbb{E}[X_i] = p_i$. Since each X_i has the same distribution and is independent of all other X_j's, we call the collection of random variables $X_1, \cdots, X_n$ a collection of *independent* and *identically distributed* (i.i.d) random variables.

Let us define a new random variable Y to be the sum of these random variables, that is, $Y = \sum_{i=1}^{n} X_i$.

1. What is the expectation of Y?
2. Show that
$$\mathrm{P}\left[Y \geq (1+\delta)\mu\right] \leq \frac{e^{\mu(e^t-1)}}{e^{(1+\delta)t\mu}}.$$
3. Show how to improve this bound even further by choosing an appropriate value of t.

The resulting best bound for $\mathrm{P}\left[Y \geq (1+\delta)\mu\right]$ is called a Chernoff bound and is an example of a *concentration* theorem - it can be shown that most of Y's probability density is concentrated about μ.

Chernoff bounds are related to the very useful *Central Limit Theorem* and also play very important roles in the analysis of randomized algorithms and the theory of machine learning. They are thus considered a cornerstone of probability theory. It is understood that everyone studying probability must have seen a Chernoff bound - now you know!

◊ Task E [4]

Another important theorem, especially important for estimation using samples: the *weak law of large numbers* (WLLN). We shall try to prove it in this task, using the Chernoff bound.

Theorem 6. *Let $X_1, \cdots, X_n$ be i.i.d random variables with each having mean μ. We define $A_n = \frac{\sum_{i=1}^{n} X_i}{n}$. Then for all $\epsilon > 0$, we have*

$$\lim_{n\to\infty} P\left[|A_n - \mu| > \epsilon\right] = 0.$$

Essentially, the average of the variables has to be roughly constant at the value μ - it takes on any other value with probability approaching 0. Intuitively, what it means is that if you keep sampling from the same distributions and add up all of them, deviations left of the mean are cancelled by deviations right of the mean, and the net result is that the sum is always roughly the same - $n\mu$ - from which it follows that the mean is always roughly μ.

Prove WLLN using the Chernoff bound from Task D. If you did not solve Task D, you may provide a proof using just the Chebyshev inequality. However, if you did solve it, you should use the Chernoff bound obtained from Task D to prove WLLN.

§ 5 A Pretty “Normal" Mixture

We have been looking at Gaussian (normal) random variables and their manipulation. Now we shall take many such Gaussians and *mix* them!

Definition 7 (GMM). *A Gaussian Mixture Model (GMM) is a random variable defined in terms of K Gaussian random variables and follows the PDF*

$$P[X = x] = \sum_{i=1}^{K} p_i P[X_i = x],$$

where each $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is a Gaussian random variable with mean μ_i and variance σ_i^2 for all $i \in \{1, 2, \ldots, K\}$. Moreover, each $p_i \geq 0$ and $\sum_{i=1}^{K} p_i = 1$.

◊ Task A [2]

To sample from a GMM's distribution, we use the following algorithm:

1. First, one of the Gaussian variables X_i is randomly chosen (or effectively, an index i is chosen) according to the PMF $\{p_1, p_2, \ldots p_k\}$. That is, i or X_i is chosen in this step with probability p_i.
2. Next, we sample a value from the chosen Gaussian's distribution $\mathcal{N}(\mu_i, \sigma_i^2)$ and this is the final value sampled from the GMM.

Suppose the output of the algorithm is random variable $\mathcal{A}$ with PDF f_{calA}, and the PDF of the GMM variable X is f_X. Show that for every $u \in \mathbb{R}$, $f_{\mathcal{A}}(u) = f_X(u)$, that is, indeed, this algorithm samples from the GMM variable's distribution.

◊ Task B [1+2+2]

Let X be a GMM sampled by the method described above where each Gaussian $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is chosen with a probability $p_i \geq 0$. Then compute

1. $\mathbb{E}[X]$.
2. $\mathrm{Var}[X]$.
3. the MGF $M_X(t)$ of X.

◊ Task C [1+1+2+2+1+1]

Now, we may be inclined to think “Isn't this just a weighted sum of Gaussians?" Let us now prove (or disprove) this property. Let us take a random variable Z to be a weighted sum of k **independent** Gaussian random variables,

$$Z = \sum_{i=1}^{K} p_i X_i,$$

where $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. For our new random variable Z, find the same expressions as in Task B:

1. $\mathbb{E}[Z]$.
2. $\mathrm{Var}[Z]$.
3. The PDF $f_Z(u)$ of Z.
4. The MGF $M_Z(t)$ of Z.
5. What can you conclude? Do X and Z have the same properties?
6. What distribution does Z seem to follow?

◊ Task D (B) [3]

Theorem 8. *For a random variable X, if it is*

1. *either finite and discrete,*
2. *or if it is continuous and its MGF $\phi_X(t)$ is known for some (non-infinitesimal) interval,*

*then its MGF and PDF **uniquely** determine each other.*

Prove the above theorem for the finite discrete case.
What can you now conclude about X and Z? Also explain logically why this may be the case.