5. Rejection Sampling

Overview

n today's lecture, we discuss applying the rejection sampling method to sample from a discrete distribution.

Learning objective:

- Understand the algorithm of rejection sampling method
- Able to implement the algorithm using Python to sample from a discrete distribution

Motivating Example

We have learned how to sample from the following distribution using the inverse transform method.

Outcomes	**Probabilities**
1	0.1
2	0.6
3	0.1
4	0.2

In this lecture, we will introduce a different method to sample from this distribution. For now, let's call this distribution our **Target Distribution**, which is the distribution we would like our final samples to be drawn from.

We have learned in the previous lecture that it is very easy to generate samples from the following distribution:

Outcomes	**Probabilities**
1	0.25
2	0.25
3	0.25
4	0.25

We are able to generate samples from this distribution easily without doing any search. [$x = \lfloor u * (b - a + 1) \rfloor + a$]. Because it is very easy to sample 1, 2, 3, 4 using the distribution above, can we first generate samples from this distribution and only keep a portion of them so that they will follow our target distribution?

An alternative idea is to do the following:

- Use a different distribution to draw samples. This distribution should allow us to very efficiently generate our samples. We call this distribution a **proposal distribution**.
- By doing the steps above, we are oversampling specific outcomes. We will only accept each sample with a specific acceptance rate. As a result, the probability of sampling each possible outcome will be equal to the target distribution we are looking for.

Strategy

Again, remember that a rejection sampling follows a two-step sampling process:

- Step 1: Each outcome will be sampled from the proposal distribution.
- Step 2: Each outcome will be kept as a final sample with a corresponding acceptance rate.

Now, we discuss how we can choose the acceptance rate for each outcome.

Choosing the probability

Without losing generality, let's assume the outcome we sampled is k. The probability of k being sampled is simply the PMF of the proposal distribution at k. Let's denote this probability using $P_p(k)$.

For Step 2, we need to compute an acceptance rate to make the probability of k of appearing among all the final samples equals $P_t(k)$, the corresponding PMF for the target distribution.

Let's explore whether using $\frac{P_t(k)}{P_p(k)}$ as the acceptance rate will work. In this way, we will have $P_p(k) \times \frac{P_t(k)}{P_p(k)} = P_t(k)$. Everything looks good except $\frac{P_t(k)}{P_p(k)}$ might be above 1. As a remedy, we might want to divide this ratio by a constant c to ensure that $\frac{P_t(k)}{P_p(k)} \leq 1$ for all k's.

If we can find such a c value, the probability of outcome k being sampled from the proposal distribution AND kept as a final sample is $P_p(k) \times \frac{P_t(k)}{P_p(k)c} = \frac{P_t(k)}{c}$.

In addition, the probability of a random sample being selected and kept as a final sample is $\frac{\sum P_t(x)}{c} = 1/c$. Thus, the conditional probability of outcome k appearing conditional on it is in the final sample is $\frac{P_t(k)}{c} / \frac{1}{c} = P_t(k)$, which is our target distribution! □ □ □

For the ideal value of c, it should be big enough so that

$$0 \leq \frac{P_t(x)}{P_p(x)c} \leq 1$$

for all the x's. At the same time, it cannot be too large, as $1/c$ represents the probability of a sample being selected as a final sample. Thus, a large c means we will be discarding lots of samples. Thus, the best c is $max(\frac{P_t(x_1)}{P_p(x_1)}, \frac{P_t(x_2)}{P_p(x_2)}, ...)$.

To sum it up, **the rejection sampling works as follows**:
- Step 1: Find out the c that gives the highest values between $\frac{P_t(k)}{P_p(k)}$ for any k.

- Step 2: Sample x from the proposal distribution

- Step 3: generate a u using np.random.rand()

- Step 4: if $u < \frac{PMF_t(x)}{PMF_p(x)c}$, we will keep the sample.
>else, we repeat Step 2 and 3, until $u < \frac{PMF_t(x)}{PMF_p(x)c}$.

Exercise 1

Target distribution:

Outcomes	Probabilities
1	0.1
2	0.6
3	0.1
4	0.2

Proposal distribution:

Outcomes	Probabilities
1	0.25
2	0.25
3	0.25
4	0.25

- Step 1: compute c using np.max(PMF_target/PMF_proposal), where PMF_proposal and PMF_target are the PMF arrays for the target distribution and the proposal distribution respectively.

- Step 2: Sample x from PMF_proposal.

- Step 3: generate a u using np.random.rand()

- Step 4: if $u < \frac{PMF_{target}(x)}{PMF_{proposal}(x)c}$, we will keep the sample.
>else, we repeat Step 2, 3, and 4 until a sample is drawn.

How to choose a good proposal distribution

- A great proposal distribution should be relatively cheap to sample from. For example, the uniform discrete distribution and geometric distributions are good distribution. To generate samples from these two distributions, we do not need to do any search.
- A great proposal distribution should follow a similar shape to the target distribution. In this way, c will be lower, which means the probability of keeping a proposal is much higher.
- A great proposal distribution should generate the same number or more outcomes compared to the target distribution. For example, if the range the target distribution is 0,1,2,3,4,5. A not so good candidate for the proposal would be a distribution that can only take 1,2,3.

Exercise 2

Use geometric distribution as the proposal distribution to generate samples from the following distribution

Outcomes	**Probabilities**
0	0.4
1	0.25
2	0.15
3	0.1
4	0.05
5	0.05