Monte Carlo Methods for Mortals

Santhosh Nadig

July 26, 2018

1 Introduction

Here's a scenario: assume that you are playing solitaire (the card game) and wondering what's the probability of a successful outcome? Well, this was a question that occurred to Stan Ulam (the Polish mathematician) who was recovering from illness and playing solitaire to while away his time. He tried to solve it using combinatorics and failed. Finally, he thought of simulating a very large number of games (with cards shuffled at random, of course) and just counting the number of successes. He, and his friend, Von Neumann, who knew a thing or two about computers and programming, set out do this calculation and eventually used this method for other scientific purposes.

This is the central idea of MC. Use randomness to "simulate" a large number of experiments. Infer / calculate a quantity of interest based on the outcome of the experiments. More the number of experiments, the better!

What are Monte Carlo (MC) methods? An (over)simplified view is that they are tools for:

1. Estimating (multi-dimensional) integrals
2. Estimating probabilities
3. ... etc.

First, a note about estimating integrals. The usual deterministic methods of numerical integration (such as trapizoidal method) suffer from what's known as the "curse of dimensionality". That is, for a fixed error the number of function evaluations (of the integrand) grows exponentially with the "dimension of integration". In general, calculations that grow exponentially are frowned upon by engineers and scientists. At the risk of giving away the story-line, it sufficies to say that MC methods do not suffer from the aforementioned "curse." I will (probably) give examples of this later on.

2 Estimating Probabilities

2.1 Random Variables

What is a random variable? As someone said, it is neither random nor variable. For example, let X denote the resistance of a resistor (assume fixed temperature and other conditions, just for argument's sake). X is an *unknown constant* (or, only known to the supreme fascist, SF). We do have, however, a series of measurements $x_1, x_2, \ldots$ from an ohm-meter. These measurements are called realizations/observations/samples of the random variable. So, now we have a random variable X and its realizations (or samples) $x_1, x_2, \ldots$ and so on.

The expected value (mean, for ordinary mortals) of a random variable, X, is defined as

$$E[X] = \int x \ p(x) \ dx,$$

where $p(x)$ is the probability density function (pdf). Assume that $p(x)$ represents a Normal distribution (or Gaussian distribution) with mean m and standard deviation σ. Thus, the probability $P(X = x) = p(x)$ is given by

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-(x-m)^2/(2\sigma^2) \Big).$$

What it means is that if you plot a histogram of $x_1, x_2, \ldots,$ it resembles the bell shape centered around m and its width $\propto \sigma$.

Assume that the unknown expected value m is to be estimated given N realizations $x_1, \ldots, x_N$ of X. We could estimate m (and hence, the integral) simply as a weighted sum of the realizations [it can proven that this is indeed the maximum likelihood estimate of m, but I will skip the derivation].

$$E[X] = m = \int x \ p(x) \ dx \approx \frac{1}{N} \sum_{i=1}^{N} x_i.$$

The estimate gets better (closer to m) as the number of samples, N, increases.

The above idea might not be a revelation. But, let us assume that we have an integral of the kind:

$$\int g(x) \cdot f(x) \ dx, \tag{1}$$

where $f(x)$ is a pdf (i.e., $f(x) \geq 0$ for all x and $\int f(x) \ dx = 1$. For instance, if $g(x) = x$, we obtain the expected value; And, if $g(x) = x^2$, we obtain the "variance", etc. These quantities are very useful - for example, the expected value represents the best estimate of the random variable X and the variance gives how confident we are of the estimate.

I1: MC method says:

$$\int g(x) \cdot f(x) \ dx \approx \frac{1}{N} \sum_i g(x'_i)$$

where $x'_1, x'_2, \ldots$ are "sampled" from the pdf $f(x)$.

2.2 Sampling

What do we mean by “sampling” from a distribution? It means drawing realizations from a given pdf. Let’s consider an example: Suppose we have a ”bent coin” which has a probability of coming up heads = 0.7 and coming up tails = 0.3. A sample is simply the result of a toss. Now, if we sampled long enough; that is, if we tossed the bent coin many times and observed the results (realizations) and plotted a histogram (normalized), then, we’d see that almost 70% of the coin tosses came up heads and the remaining 30% or so came up tails. A Python script to simulate such a coin is shown below:

```
"""
Simlate a biased coin
"""

import numpy as np

class biased_coin:
    """ biased_ is a class that implements a biased coin with
        probability of heads = p, probability of tails = 1-p """
    def __init__(self, p):
        self.p = p

    def toss(self):
        """ generate a single toss of the biased coin """
        if np.random.rand() > self.p:
            return 'T'
        else:
            return 'H'

    def get_sequence(self,num_tosses):

        seq = [self.toss() for i in range(num_tosses)]
        return seq


def main():
    p = 0.7 # probability of heads
    c = biased_coin(0.7);
    tosses = c.get_sequence(100);
    num_heads = tosses.count('H')
    num_tails = tosses.count('T')
    print ("simulated 100 tosses of a biased coin with p = " + str(p))
    print ("obtained " + str(num_heads) + " heads and " + str(num_tails) + " tails")



if __name__ == '__main__':
    main()


----- Sample Output -----
simulated 100 tosses of a biased coin with p = 0.7
obtained 71 heads and 29 tails
```

A key observation in the above program is the following: we first sample from a uniform distribution (a random number that has the same probability of assuming any value between 0 and 1.0). This is provided by the function `np.random.rand()`. This random variable is then used to determine whether the appropriate result of the coin toss shall be heads or tails. If the random number is $> p(= 0.7)$, then the outcome is tails; otherwise heads. So, about 70% of the coin tosses result in heads as expected.

I2: Random variables from a uniform distribution are "translated" to another desired distribution.

This translation is a key feature of MC methods. There are a plethora of methods that achieve this (e.g., inverse transform sampling, rejection sampling, etc etc). But, the basic idea is that random variables from one distribution is used to generate random variables of another (desired) distribution.

The above example had a discrete pdf (a probabilty mass function (pmf)). If we assume a continuous pdf (such as a Normal distribution), then, the histogram of the samples would resemble a "bell curve".

2.3 Recap

Just a quick recap; the first idea (**I1**) says that integrals of the kind (1) can be approximated by averaging $g(x)$ s evaluated at xs drawn from the pdf $f(x)$. The second idea (**I2**) says that samples from one distribution (such as uniform distribution, which is provided as a standard library in most OSes) can be used to generate samples from another distribution.

2.4 Conditional Probabilty

In this section, we explore how to generate samples from a pdf as a function of other pdfs. I will use the term probability density function (which, if you remember, is applicable to a continuous random variable) instead of probability mass function (of a discrete random variable) just for convinience. I know that this violates mathematical rigour, but it is worth the sacrifice (in my opinion).

2.4.1 Task 1

Assume that we are given samples from a pdf $p(x)$ and we know the conditional pdf $p(y|x)$. Our task is to generate samples from the pdf $p(y)$. So, say that we have samples $x_1, x_2, \ldots, x_N$ from the pdf:

$$p(x) = \begin{cases} 0.6, & x = 0. \\ 0.4, & x = 1. \end{cases} \tag{2}$$

We are also given the conditional probabilities:

$$p(y|x = 0) = \begin{cases} 0.1, & y = 0. \\ 0.9, & y = 1. \end{cases} \tag{3}$$

$$p(y|x = 1) = \begin{cases} 0.7, & y = 0. \\ 0.3, & y = 1. \end{cases} \tag{4}$$

[1] Now, the first task is to draw samples from the distribution $p(y)$. One way to approach this task is to estimate $p(x)$ (call it $\hat{p}(x)$) using the samples $x_1, \ldots, x_N$ and then apply a simple formula to calculate $p(y)$ as:

$$p(y = 0) = p(y = 0|x = 0) \cdot \hat{p}(x = 0) + p(y = 0|x = 1) \cdot \hat{p}(x = 1) \tag{5}$$
$$p(y = 1) = p(y = 1|x = 0) \cdot \hat{p}(x = 0) + p(y = 1|x = 1) \cdot \hat{p}(x = 1) \tag{6}$$

and then, sample from $p(y)$ as before. Another approach is to do the following: For each x_i, simulate M samples of ys (i.e., $y_1, y_2, \ldots, y_M$) using the conditional probability $p(y|x_i)$. Thus, we end up with $M \cdot N$ samples of y which are essentially distributed as $p(y)$. The following code sample illustrates the procedure (with $N = 10000$ and $M = 10$).

[1] Digression: Althought this is an artificial toy example, it bears resemblance to a binary assymetric channel where y can be thought of as the received signal.

```
  x = sample(x_labels, p_x, 10000)
  y = np.array([])
  for i in range(len(x)):
      q = p_y_x[x[i],]
      y = np.append(y, sample(y_labels, q, 10))

  cnt_y = np.array([(y == 0).sum(), (y == 1).sum()])
  p_y = cnt_y/len(y)
  print('p(y) is: [%1.2f, %1.2f]'%(p_y[0], p_y[1]))

----- Sample Output -----
p(y) is: [0.34, 0.66]
```

2.4.2 Task 2

For the next task, let us suppose the following:

1. The pdf $p(x)$ (a.k.a prior) is known
2. The conditional pdfs $p(y|x)$ (a.k.a likelihood) is known
3. We have a y_i, which is either a 0 or 1 in our case (a.k.a measurement); and
4. **Our job is to find the best estimate of x that might have generated the given y_i. That is, we need to estimate the posterior probability $p(x|y)$.**

In theory, this can be achieved by applying Bayes' theorem

$$\textbf{posterior} = \frac{\textbf{prior} \cdot \textbf{likelihood}}{\textbf{evidence}} \tag{7}$$

$$p(x = x_j|y = y_i) = \frac{p(y = y_i|x = x_j)p(x = x_j)}{\sum_j p(y = y_i|x = x_j)p(x = x_j)} \tag{8}$$

Say, $y_i = 0$ and we would like to determine $p(x|y = 0)$. Using the above theorem, it turns out to be:

$$p(x = 0|y = 0) = \frac{p(y = 0|x = 0)p(x = 0)}{p(y = 0|x = 0)p(x = 0) + p(y = 0|x = 1)p(x = 1)} = 0.176$$

$$p(x = 1|y = 0) = (1 - 0.176) = 0.823. \tag{9}$$

One approach to solve this problem the "Monte Carlo" way is to do the following:

1. Generate "candidate" samples of x (say, $x_1, \ldots, x_N$) from the known pdf $p(x)$.
2. For each of the x_is generated, generate a "measurement" (call it $\hat{y}$) according to the pdf (likelihood) $p(y|x)$.
3. If $\hat{y}$ is the same as y, save that x_i, else discard the x_i.

Once the above steps are complete, we would have a new (but reduced) set of x_is (say, $x_1, \ldots, x_M$) which would be distributed according to the pdf $p(x|y)$. The following code sample illustrates the procedure.

```
 x = sample(x_labels, p_x, 10000)
    x_new = np.array([])
    for i in range(len(x)):
        q = p_y_x[x[i],]
        y = sample(y_labels, q, 1)
        if y == 0:
            x_new = np.append(x_new, x[i])

    #done! now, check the distribution
    cnt_x_new = np.array([(x_new == 0).sum(), (x_new == 1).sum()])
    p_x_y0 = cnt_x_new / len(x_new)
    print('p(x|y=0) is: [%1.2f, %1.2f]'%(p_x_y0[0], p_x_y0[1]))
    print('INFO: len(x) = %d, len(x_new) = %d'%(len(x), len(x_new)))
```

```
----- Sample Output -----
p(x|y=0) is: [0.18, 0.82]
INFO: len(x) = 10000, len(x_new) = 3375
```

2.5 Recap

We have seen one way of generating samples from conditional probabilities (or sampling from the posterior probability). The method in the example we used discarded samples from a set of candidates (generated using the prior pdf) to generate another set of samples which were distributed according to the posterior pdf. If this method is applied recursively, one may end up with too few (or even no) samples. However, there are ways to overcome this problem.