What is the Average Value as the Number of Samples Increases

In this lesson, we will learn how to calculate the average value as the number of samples increases.

WE'LL COVER THE FOLLOWING

In the previous lesson, we reviewed the meaning of “expected value”: when you get a whole bunch of samples from a distribution, and a function on those samples, what is the average value of the function’s value as the number of samples gets large?

Revisiting the Naive Implementation of Expected Value

We gave a naive implementation:

```
public static double ExpectedValue<T>(
    this IDistribution<T> d,
    Func<T, double> f) =>
  d.Samples().Take(1000).Select(f).Average();

public static double ExpectedValue(
    this IDistribution<double> d) =>
  d.ExpectedValue(x=>x);
```

Issues with the Naive Approach

Though short and sweet, this implementation has some problems; the most obvious one is that hard-coded 1000 in there; where did it come from? Nowhere, in particular, that's where.

It seems highly likely that this is either too big, and we can compute a reasonable estimate in fewer samples, or that it is too small, and we're missing some important values.

Let's explore a scenario where the number of samples is too small. The scenario will be contrived but not entirely unrealistic.

Let's suppose we have an investment strategy; we invest a certain amount of money with this strategy, and when the strategy is complete, we have either more or less money than we started with. To simplify things, let's say that the "outcome" of this strategy is just a number between 0.0 and 1.0; 0.0 indicates that the strategy has completely failed, resulting in a loss, and 1.0 indicates that the strategy has completely succeeded, resulting in the maximum possible return.

Before we go on, we want to talk a bit about that "resulting in a loss" part. If you're a normal, sensible investor and you have $100 to invest, you buy a stock or a mutual fund for $100 because you believe it will increase in value. If it goes up to $110, you sell it and pocket the $10 profit. If it goes down to $90, you sell it and take the $10 loss. But in no case do you ever lose more than the $100 you invested. (Though of course, you do pay fees on each trade whether it goes up or down; let's suppose those are negligible.) Our goal is to get that 10% return on investment.

Now consider the following much riskier strategy for spending $100 to speculate in the market: suppose there is a stock currently at $100 which we believe will go down, not up. We borrow a hundred shares from someone willing to lend them to us and sell them for $10000. We pay the lender $100 interest for their trouble. Now if the stock goes down to $90, we repurchase a hundred shares for $9000, return them to the owner, and we've spent $100 but received $1000; we've made a 900% return on the $100 we "invested". This is a "short sale", and as you can see, you get a 900% return instead of a 10% return.

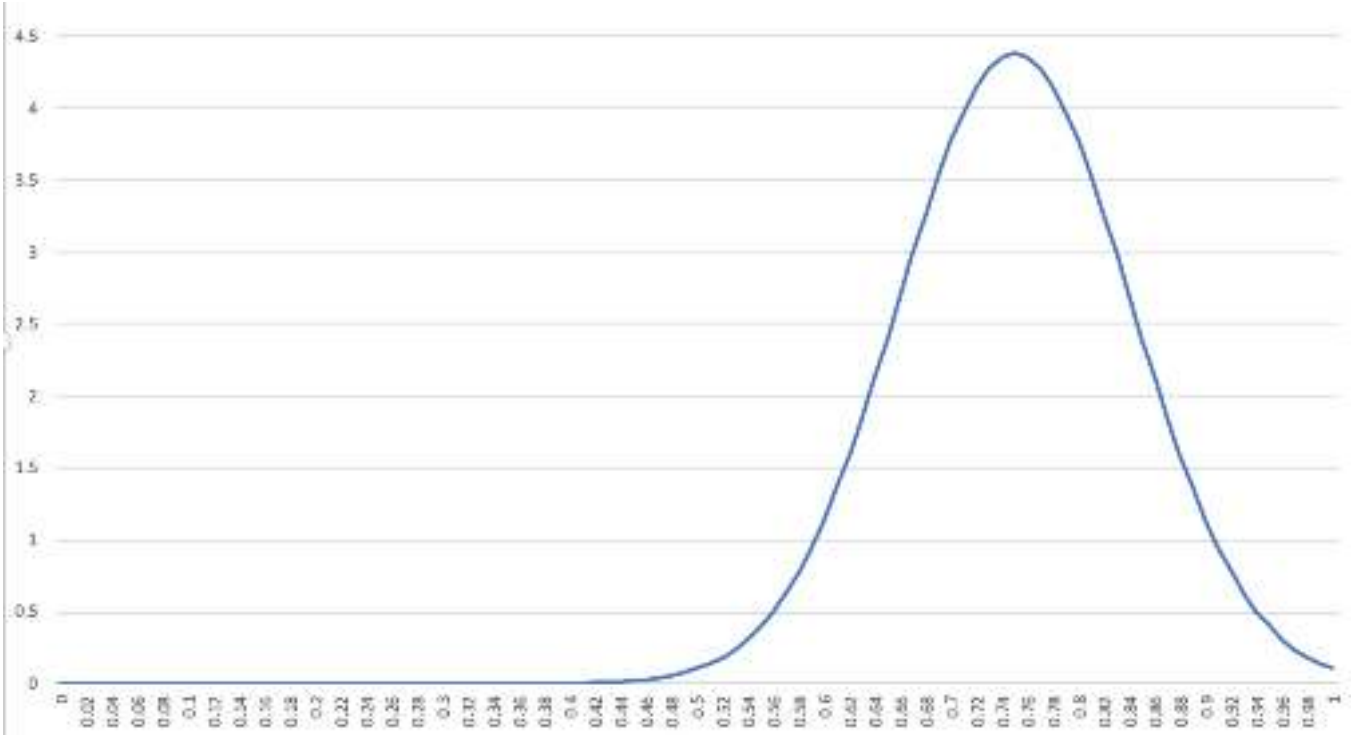
> (We say “invested” in scare quotes because this isn’t investing; it’s speculation. Which is a genteel word for “gambling”.)

But perhaps you also see the danger here. Suppose the stock goes down but only to $99.50. We buy back the shares for $9950, and we’ve only gained $50 on the trade, so we’ve lost half of our $100; in the “long” strategy, we would only have lost fifty cents; your losses can easily be bigger with the short strategy than with the long strategy.

But worse, what if the stock goes **up to** $110. We have to buy back those shares for $11000, so we’ve “invested” $100 and gotten a “return” of −$1000, for a total loss of $1100 on a $100 investment. In a short sale if things go catastrophically wrong, you can end up losing more than your original investment. A lot more!

As we often say, foreshadowing is the sign of a quality blog; let’s continue with our example.

We have a process that produces values from 0.0 to 1.0 that indicates the “success level” of the strategy. What is the distribution of success level? Let’s suppose it’s a straightforward bell-shaped curve with the mean on the “success” side:



This is a PDF, and extra bonus, it is normalized: the total area under the curve is 1.0. The vast majority of the time — 99.9% — our investment strategy produces an outcome between 0.48 and 1.0, so that's the area of the graph we are going to focus on.

> We know it looks a little weird seeing a bell-shaped curve that just cuts off abruptly at 1.0, but let's not stress about it. Again, this is a contrived example, and we'll see why it is irrelevant later.

Now, we don't intend to imply that the "success level" is the return: we are not saying that most of the time we get a 75% return. Let's suppose that we've worked out a function that translates "strategy outcome level" to the percentage gain on our investment. That is, if the function is 0.0, we've not gained anything but we've not lost anything either. If it is 0.5 then our profits are 50% of our investment; if it's -0.25 then we've taken a net loss of 25% of our investment, and so on.

Let's propose such a function, and zoom in on the right side of the diagram where the 99.9% of cases are found: from 0.46 to 1.0. Here's our function:

If our process produces an outcome of 0.54 or larger, we make between a 0%

and an 18% return, which seems pretty good; after all the vast majority of the bulk of the distribution is to the right of 0.54. Now, in the unlikely case where we get an outcome from 0.48 to 0.54, we lose money; on the left-hand end, we're losing almost 200%; that is, if we put in $100 we not only fail to make it back, we end up owing $100.

What is the Average Value of the "Profit Function"?

The question at hand, is what is the average value of the "profit function" given the distribution of outcomes? That is, if we ran this strategy a thousand times, on average what return would we get?

Let's have a look at the code:

```
var p = Normal.Distribution(0.75, 0.09);
Func<double, double> f = x =>
  Atan(1000 * (x - .45)) * 20 - 31.2;
Console.WriteLine($"{p.ExpectedValue(f):0.##}");
```

> We are ignoring the fact that if we get a sample out of the normal distribution that is greater than 1.0 or less than 0.0 we do not discard it. The region to the left of 0.0 is absurdly unlikely, and the region to the right of 1.0 has low probability and the value of the profit function is very flat.

Also, we are limiting the distribution to support of 0.0 to 1.0 for pedagogical reasons; as we'll see in later lessons, we will eventually remove this restriction, so let's just ignore it now.

We get the answer:

```
0.14
```

Super, we will get on average a 14% return on our investment with this strategy. Seems like a good investment; though we have a small chance of losing big, the much larger chance of getting a solid 0% to 18% return outweighs it. On average.

> It is important to realize that this 14% figure does not necessarily imply

> It is important to realize that this 14% figure does not necessarily imply that typically we get a 14% return when we run this strategy, any more
>
> than the expected value of 3.5 implies that we'll typically get 3.5 when we roll a $d6$. The expected value only tells you what the average behavior is; it doesn't tell you, for example, what the probability is that you'll lose everything. It tells you that if you ran this strategy a million times, on average you'd get a 14% return, and that's all it tells you. Be careful when making decisions relying upon expected values!

This 14% expected result totally makes sense, right? We know most of the time the result will be between 0% and 18%, so 14% seems like a totally reasonable expected value.

Let's just run the code again to be sure.

```
0.14
```

Whew. That's a relief.

You know what they say: doing the same thing twice and expecting different results is the definition of practicing the piano, but we are still not satisfied. Let's run it another hundred times:

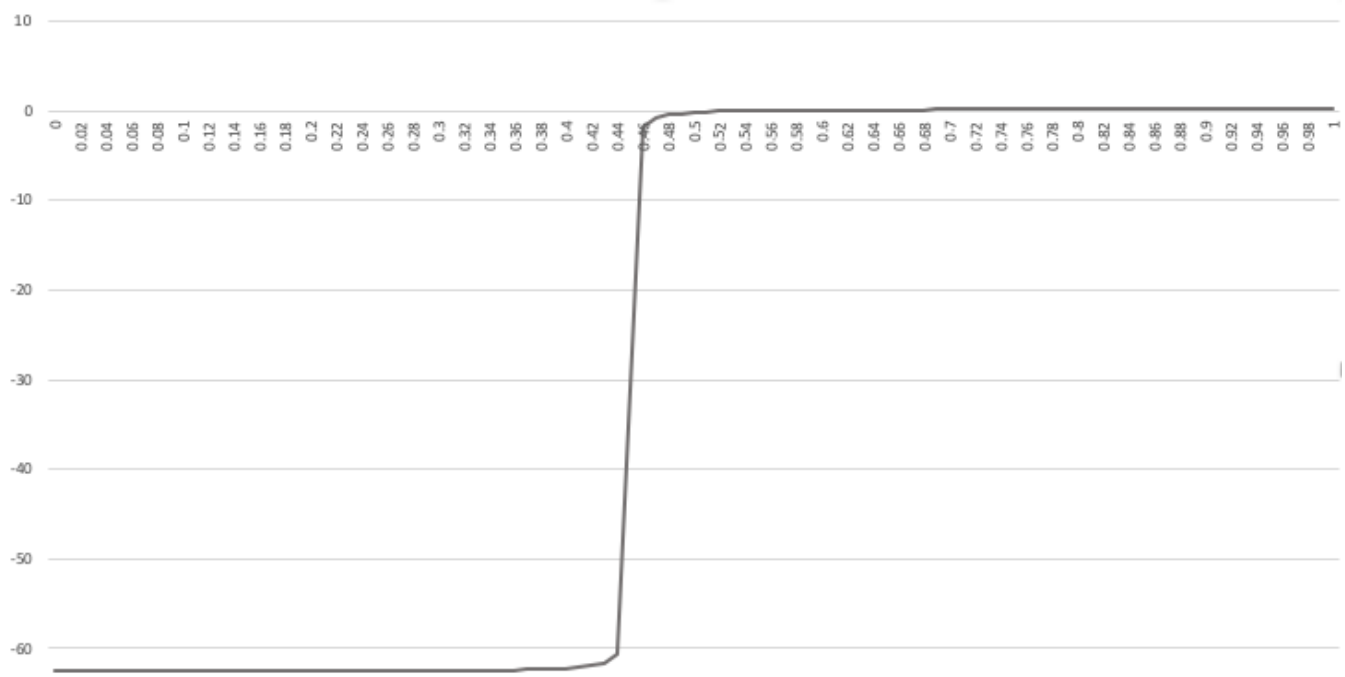
```
0.14, 0.08, 0.09, 0.14, 0.08, 0.14, 0.14, 0.07, 0.07, 0.14, ...
```

Oh dear me.

The expected return of our strategy is usually 14%; why it is sometimes half that? Yes, there is randomness in our algorithm, but surely it should not cause the value computed to vary so widely!

It seems like we cannot rely on this implementation, but why not? The code looks right.

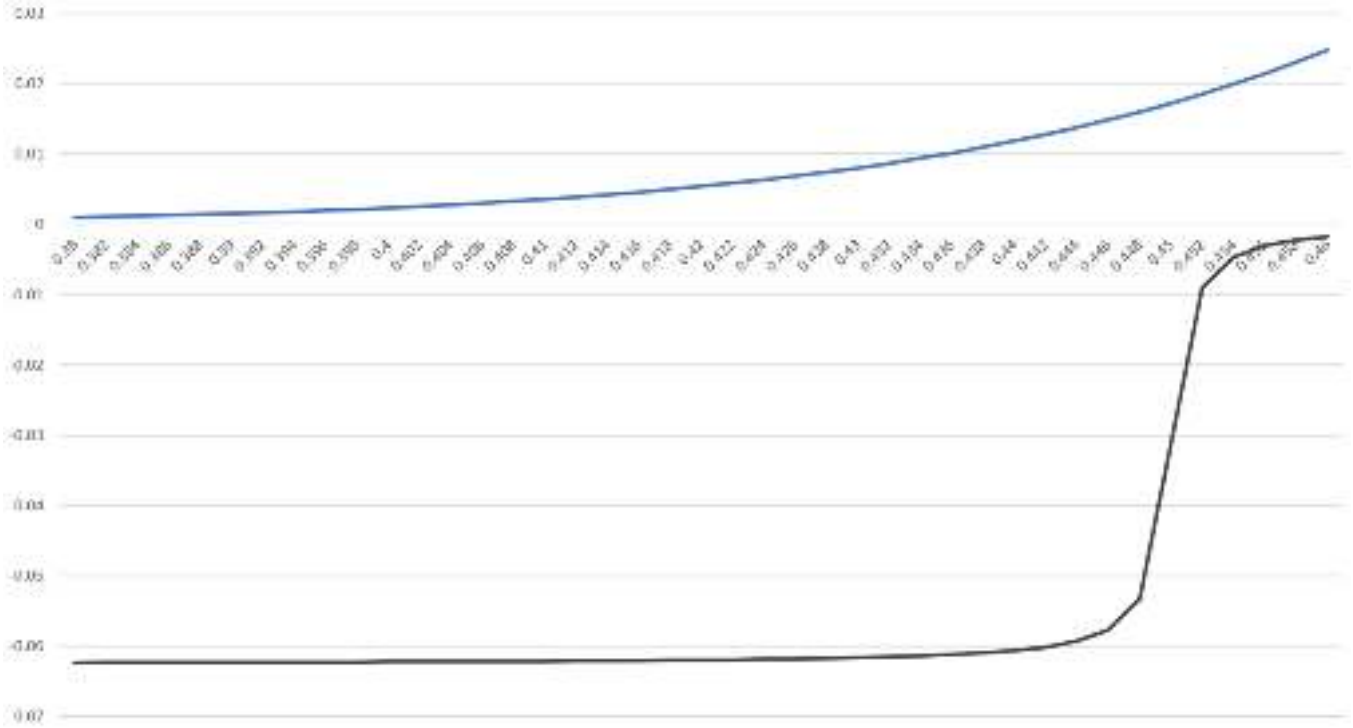
As it turns out, we showed you exactly the wrong part of the graphs above. We showed you the parts that makeup 99.9% of the likely cases. Let's look at the graph of the profit-or-loss function over the whole range of outcomes from 0.0 to 1.0:



It's an approximation of a step function; in the original graph that started at 0.46, we should have noticed that the extremely steep part of the curve was trending downwards to the left at an alarming rate. If you have an outcome of 0.48 from our process, you're going to lose half your money; 0.46, you lose all your money and a little bit more. But by the time we get to 0.44, we're down to losing over 60 times your original investment; this is a catastrophic outcome.

How likely is that catastrophic outcome?

Let's zoom in on just that part of the probability distribution. We'll re-scale the profit function so that they fit on the same graph, so you can see where exactly the step happens:



This is a normalized PDF, so the probability of getting a particular outcome is equal to the area of the region under the curve. The region under the curve from 0.38 to 0.45 is a little less than a triangle with base 0.07 and height 0.02, so its area is in the ballpark of 0.0007. We will generate a sample in that region roughly once every 1400 samples.

But... we're computing the expected value by taking only a thousand samples, so it is quite likely that we don't hit this region at all. The expected losses in that tiny area are so enormous that it changes the average every time we do sample from it. Whether you include a -60 value or not in the average is a huge influence as to whether the average is 0.14 or 0.07.

> If you run the expected value estimate a few thousand times or more it just gets crazier; sometimes the expected value is actually negative if we just by luck get more than one sample in this critical region.

We contrived this example to produce occasional "black swan events"; obviously, our naïve algorithm for computing expected value does not take into account the difficulty of sampling a black swan event, and therefore produces inaccurate and inconsistent results.

Implementation

Well, we already wrote the code, so let's run it:

Program.cs

Beta.cs

Bernoulli.cs

BetterRandom.cs

Distribution.cs

DistributionBuilder.cs

Empty.cs

Episode32.cs

Extensions.cs

Flip.cs

Gamma.cs

IDiscreteDistribution.cs

IDistribution.cs

IWeightedDistribution.cs

Markov.cs

```
using System;
using static System.Math;
namespace Probability
{
    static class Episode32
    {
        public static void DoIt()
        {
            Console.WriteLine("Episode 32 -- Black swan attack");

            var p = Normal.Distribution(0.75, 0.09);
            Func<double, double> f = x => Atan(1000 * (x - .45)) * 20 - 31.2;
            for (int i = 0; i < 100; ++i)
                Console.WriteLine($"{p.ExpectedValue(f):0.##}");
        }
    }
}
```

We know how to solve this problem exactly for discrete distributions; can we use some of those ideas to improve our estimate of the expected value in this scenario?