Chapter 8

Estimation

8.1 The estimation game

Let's play a game. I'll think of a distribution, and you have to guess what it is. We'll start out easy and work our way up.

I'm thinking of a distribution. I'll give you two hints; it's a normal distribution, and here's a random sample drawn from it:

$\{-0.441, 1.774, -0.101, -1.138, 2.975, -2.138\}$

What do you think is the mean parameter, μ, of this distribution?

One choice is to use the sample mean to estimate μ. Up until now we have used the symbol μ for both the sample mean and the mean parameter, but now to distinguish them I will use $\bar{x}$ for the sample mean. In this example, $\bar{x}$ is 0.155, so it would be reasonable to guess $\mu = 0.155$.

This process is called **estimation**, and the statistic we used (the sample mean) is called an **estimator**.

Using the sample mean to estimate μ is so obvious that it is hard to imagine a reasonable alternative. But suppose we change the game by introducing outliers.

I'm thinking of a distribution. It's a normal distribution, and here's a sample that was collected by an unreliable surveyor who occasionally puts the decimal point in the wrong place.

$\{-0.441, 1.774, -0.101, -1.138, 2.975, -213.8\}$

Now what's your estimate of μ? If you use the sample mean your guess is -35.12. Is that the best choice? What are the alternatives?

One option is to identify and discard outliers, then compute the sample mean of the rest. Another option is to use the median as an estimator.

Which estimator is the best depends on the circumstances (for example, whether there are outliers) and on what the goal is. Are you trying to minimize errors, or maximize your chance of getting the right answer?

If there are no outliers, the sample mean minimizes the **mean squared error** (MSE). If we play the game many times, and each time compute the error $\bar{x} - \mu$, the sample mean minimizes

$$MSE = \frac{1}{m} \sum (\bar{x} - \mu)^2$$

Where m is the number of times you play the estimation game (not to be confused with n, which is the size of the sample used to compute $\bar{x}$).

Minimizing MSE is a nice property, but it's not always the best strategy. For example, suppose we are estimating the distribution of wind speeds at a building site. If we guess too high, we might overbuild the structure, increasing its cost. But if we guess too low, the building might collapse. Because cost as a function of error is asymmetric, minimizing MSE is not the best strategy.

As another example, suppose I roll three six-sided dice and ask you to predict the total. If you get it exactly right, you get a prize; otherwise you get nothing. In this case the value that minimizes MSE is 10.5, but that would be a terrible guess. For this game, you want an estimator that has the highest chance of being right, which is a **maximum likelihood estimator** (MLE). If you pick 10 or 11, your chance of winning is 1 in 8, and that's the best you can do.

Exercise 8.1 Write a function that draws 6 values from a normal distribution with $\mu = 0$ and $\sigma = 1$. Use the sample mean to estimate μ and compute the error $\bar{x} - \mu$. Run the function 1000 times and compute MSE.

Now modify the program to use the median as an estimator. Compute MSE again and compare to the MSE for $\bar{x}$.

8.2 Guess the variance

I'm thinking of a distribution. It's a normal distribution, and here's a (familiar)

sample:

$\{-0.441, 1.774, -0.101, -1.138, 2.975, -2.138\}$

What do you think is the variance, σ^2, of my distribution? Again, the obvious choice is to use the sample variance as an estimator. I will use $S^2$to denote the sample variance, to distinguish from the unknown parameter σ^2.

$$S^2 = \frac{1}{n}\sum(x_i - \bar{x})^2$$

For large samples, $S^2$is an adequate estimator, but for small samples it tends to be too low. Because of this unfortunate property, it is called a **biased** estimator.

An estimator is **unbiased** if the expected total (or mean) error, after many iterations of the estimation game, is 0. Fortunately, there is another simple statistic that is an unbiased estimator of σ^2:

$$S^2_{n-1} = \frac{1}{n-1}\sum(x_i - \bar{x})^2$$

The biggest problem with this estimator is that its name and symbol are used inconsistently. The name "sample variance" can refer to either S^2or S^2_{n-1}, and the symbol $S^2$is used for either or both.

For an explanation of why $S^2$is biased, and a proof that $S^2_{n-1}$is unbiased, see `http://wikipedia.org/wiki/Bias_of_an_estimator`.

Exercise 8.2 Write a function that draws 6 values from a normal distribution with $\mu = 0$ and $\sigma = 1$. Use the sample variance to estimate σ^2 and compute the error $S^2 - \sigma^2$. Run the function 1000 times and compute mean error (not squared).

Now modify the program to use the unbiased estimator S^2_{n-1}. Compute the mean error again and see if it converges to zero as you increase the number of games.

8.3 Understanding errors

Before we go on, let's clear up a common source of confusion. Properties like MSE and bias are long-term expectations based on many iterations of the estimation game.

While you are playing the game, you don't know the errors. That is, if I give you a sample and ask you to estimate a parameter, you can compute the value of the estimator, but you can't compute the error. If you could, you wouldn't need the estimator!

The reason we talk about estimation error is to describe the behavior of different estimators in the long run. In this chapter we run experiments to examine those behaviors; these experiments are artificial in the sense that we know the actual values of the parameters, so we can compute errors. But when you work with real data, you don't, so you can't.

Now let's get back to the game.

8.4 Exponential distributions

I'm thinking of a distribution. It's an exponential distribution, and here's a sample:

{5.384, 4.493, 19.198, 2.790, 6.122, 12.844}

What do you think is the parameter, λ, of this distribution?

In general, the mean of an exponential distribution is $1/\lambda$, so working backwards, we might choose

$$\hat{\lambda} = 1 \ / \ \bar{x}$$

It is common to use hat notation for estimators, so $\hat{\lambda}$ is an estimator of λ. And not just any estimator; it is also the MLE estimator[1]. So if you want to maximize your chance of guessing λ exactly, $\hat{\lambda}$ is the way to go.

But we know that $\bar{x}$ is not robust in the presence of outliers, so we expect $\hat{\lambda}$ to have the same problem.

Maybe we can find an alternative based on the sample median. Remember that the median of an exponential distribution is ln(2) / λ, so working backwards again, we can define an estimator

$$\hat{\lambda}_{1/2} = \ln(2) / \mu_{1/2}$$

where $\mu_{1/2}$ is the sample median.

Exercise 8.3 Run an experiment to see which of $\hat{\lambda}$ and $\hat{\lambda}_{1/2}$ yields lower MSE. Test whether either of them is biased.

[1] See `http://wikipedia.org/wiki/Exponential_distribution#Maximum_likelihood`.

8.5 Confidence intervals

So far we have looked at estimators that generate single values, known as **point estimates**. For many problems, we might prefer an interval that specifies an upper and lower bound on the unknown parameter.

Or, more generally, we might want that whole distribution; that is, the range of values the parameter could have, and for each value in the range, a notion of how likely it is.

Let's start with **confidence intervals**.

I'm thinking of a distribution. It's an exponential distribution, and here's a sample:

{5.384, 4.493, 19.198, 2.790, 6.122, 12.844}

I want you to give me a range of values that you think is likely to contain the unknown parameter λ. More specifically, I want a 90% confidence interval, which means that if we play this game over and over, your interval will contain λ 90% of the time.

It turns out that this version of the game is hard, so I'm going to tell you the answer, and all you have to do is test it.

Confidence intervals are usually described in terms of the miss rate, α, so a 90% confidence interval has miss rate $\alpha = 0.1$. The confidence interval for the λ parameter of an exponential distribution is

$$\left(\hat{\lambda} \frac{\chi^2(2n, 1-\alpha/2)}{2n}, \hat{\lambda} \frac{\chi^2(2n, \alpha/2)}{2n} \right)$$

where n is the sample size, $\hat{\lambda}$ is the mean-based estimator from the previous section, and $\chi^2(k, x)$ is the CDF of a chi-squared distribution with k degrees of freedom, evaluated at x (see `http://wikipedia.org/wiki/Chi-square_distribution`).

In general, confidence intervals are hard to compute analytically, but relatively easy to estimate using simulation. But first we need to talk about Bayesian estimation.

8.6 Bayesian estimation

If you collect a sample and compute a 90% confidence interval, it is tempting to say that the true value of the parameter has a 90% chance of falling in the

interval. But from a frequentist point of view, that is not correct because the parameter is an unknown but fixed value. It is either in the interval you computed or not, so the frequentist definition of probability doesn't apply.

So let's try a different version of the game.

I'm thinking of a distribution. It's an exponential distribution, and I chose λ from a uniform distribution between 0.5 and 1.5. Here's a sample, which I'll call X:

{2.675, 0.198, 1.152, 0.787, 2.717, 4.269}

Based on this sample, what value of λ do you think I chose?

In this version of the game, λ *is* a random quantity, so we can reasonably talk about its distribution, and we can compute it easily using Bayes's theorem.

Here are the steps:

1. Divide the range (0.5, 1.5) into a set of equal-sized bins. For each bin, we define H_i, which is the hypothesis that the actual value of λ falls in the i th bin. Since λ was drawn from a uniform distribution, the prior probability, $P(H_i)$, is the same for all i.

2. For each hypothesis, we compute the likelihood, $P(X \mid H_i)$, which is the chance of drawing the sample X given H_i.

$$P(X \mid H_i) = \prod_j \text{expo}(\lambda_i, x_j)$$

   where expo(λ, x) is a function that computes the PDF of the exponential distribution with parameter λ, evaluated at x.

$$PDF_{expo}(\lambda, x) = \lambda e^{-\lambda x}$$

   The symbol $\prod$ represents the product of a sequence (see `http://wikipedia.org/wiki/Multiplication#Capital_Pi_notation`).

3. Then by Bayes's theorem the posterior distribution is

   $P(H_i \mid X) = P(H_i)\ P(X \mid H_i)\ / f$

   where f is the normalization factor

$$f = \sum_i P(H_i) P(X \mid H_i)$$

Given a posterior distribution, it is easy to compute a confidence interval. For example, to compute a 90% CI, you can use the 5th and 95th percentiles of the posterior.

Bayesian confidence intervals are sometimes called **credible intervals**; for a discussion of the differences, see `http://wikipedia.org/wiki/Credible_interval`.

8.7 Implementing Bayesian estimation

To represent the prior distribution, we could use a Pmf, Cdf, or any other representation of a distribution, but since we want to map from a hypothesis to a probability, a Pmf is a natural choice.

Each value in the Pmf represents a hypothesis; for example, the value 0.5 represents the hypothesis that λ is 0.5. In the prior distribution, all hypotheses have the same probability. So we can construct the prior like this:

```
def MakeUniformSuite(low, high, steps):
    hypos = [low + (high-low) * i / (steps-1.0) for i in range(steps)]
    pmf = Pmf.MakePmfFromList(hypos)
    return pmf
```

This function makes and returns a Pmf that represents a collection of related hypotheses, called a **suite**. Each hypothesis has the same probability, so the distribution is **uniform**.

The arguments `low` and `high` specify the range of values; `steps` is the number of hypotheses.

To perform the update, we take a suite of hypotheses and a body of evidence:

```
def Update(suite, evidence):
    for hypo in suite.Values():
        likelihood = Likelihood(evidence, hypo)
        suite.Mult(hypo, likelihood)
    suite.Normalize()
```

For each hypothesis in the suite, we multiply the prior probability by the likelihood of the evidence. Then we normalize the suite.

In this function, `suite` has to be a Pmf, but `evidence` can be any type, as long as `Likelihood` knows how to interpret it.

Here's the likelihood function:

```
def Likelihood(evidence, hypo):
    param = hypo
    likelihood = 1
    for x in evidence:
        likelihood *= ExpoPdf(x, param)

    return likelihood
```

In `Likelihood` we assume that `evidence` is a sample from an exponential distribution and compute the product in the previous section.

`ExpoPdf` evaluates the PDF of the exponential distribution at `x`:

```
def ExpoPdf(x, param):
    p = param * math.exp(-param * x)
    return p
```

Putting it all together, here's the code that creates the prior and computes the posterior:

```
evidence = [2.675, 0.198, 1.152, 0.787, 2.717, 4.269]
prior = MakeUniformSuite(0.5, 1.5, 100)
posterior = prior.Copy()
Update(posterior, evidence)
```

You can download the code in this section from `http://thinkstats.com/estimate.py`.

When I think of Bayesian estimation, I imagine a room full of people, where each person has a different guess about whatever you are trying to estimate. So in this example they each have a guess about the correct value of λ.

Initially, each person has a degree of confidence about their own hypothesis. After seeing the evidence, each person updates their confidence based on $P(E \mid H)$, the likelihood of the evidence, given their hypothesis.

Most often the likelihood function computes a probability, which is at most 1, so initially everyone's confidence goes down (or stays the same). But then we normalize, which increases everyone's confidence.

So the net effect is that some people get more confident, and some less, depending on the relative likelihood of their hypothesis.

8.8 Censored data

The following problem appears in Chapter 3 of David MacKay's *Information Theory, Inference and Learning Algorithms*, which you can download from `http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/`.

> Unstable particles are emitted from a source and decay at a distance x, a real number that has an exponential probability distribution with [parameter] λ. Decay events can only be observed if they occur in a window extending from $x = 1$ cm to $x = 20$ cm. n decays are observed at locations $\{ x_1, \ldots, x_N \}$. What is λ?

This is an example of an estimation problem with **censored data**; that is, we know that some data are systematically excluded.

One of the strengths of Bayesian estimation is that it can deal with censored data with relative ease. We can use the method from the previous section with only one change: we have to replace PDF_{expo} with the conditional distribution:

$$\text{PDF}_{cond}(\lambda, x) = \lambda e^{-\lambda x} / Z(\lambda)$$

for $1 < x < 20$, and 0 otherwise, with

$$Z(\lambda) = \int_1^{20} \lambda e^{-\lambda x}\, dx = e^{-\lambda} - e^{-20\lambda}$$

You might remember $Z(\lambda)$ from Exercise 6.5. I told you to keep it handy.

Exercise 8.4 Download `http://thinkstats.com/estimate.py`, which contains the code from the previous section, and make a copy named `decay.py`.

Modify `decay.py` to compute the posterior distribution of λ for the sample $X = \{1.5, 2, 3, 4, 5, 12\}$. For the prior you can use a uniform distribution between 0 and 1.5 (not including 0).

You can download a solution to this problem from `http://thinkstats.com/decay.py`.

Exercise 8.5 In the 2008 Minnesota Senate race the final vote count was 1,212,629 votes for Al Franken and 1,212,317 votes for Norm Coleman. Franken was declared the winner, but as Charles Seife points out in *Proofiness*, the margin of victory was much smaller than the margin of error, so the result should have been considered a tie.

Assuming that there is a chance that any vote might be lost and a chance that any vote might be double-counted, what is the probability that Coleman actually received more votes?

Hint: you will have to fill in some details to model the error process.

8.9 The locomotive problem

The locomotive problem is a classic estimation problem also known as the "German tank problem." Here is the version that appears in Mosteller, *Fifty Challenging Problems in Probability*:

> "A railroad numbers its locomotives in order 1..N. One day you see a locomotive with the number 60. Estimate how many locomotives the railroad has."

Before you read the rest of this section, try to answer these questions:

1. For a given estimate, $\hat{N}$, what is the likelihood of the evidence, $P(E \mid \hat{N})$? What is the maximum likelihood estimator?

2. If we see train i it seems reasonable that we would guess some multiple of i so let's assume $\hat{N} = ai$. What value of a minimizes mean squared error?

3. Still assuming that $\hat{N} = ai$ can you find a value of a that makes $\hat{N}$ an unbiased estimator?

4. For what value of N is 60 the average value?

5. What is the Bayesian posterior distribution assuming a prior distribution that is uniform from 1 to 200?

For best results, you should take some time to work on these questions before you continue.

For a given estimate, $\hat{N}$, the likelihood of seeing train i is $1/\hat{N}$ if $i \leq \hat{N}$, and 0 otherwise. So the MLE is $\hat{N} = i$. In other words, if you see train 60 and you want to maximize your chance of getting the answer exactly right, you should guess that there are 60 trains.

But this estimator doesn't do very well in terms of MSE. We can do better by choosing $\hat{N} = ai$; all we have to do is find a good value for a.

Suppose that there are, in fact, N trains. Each time we play the estimation game, we see train i and guess ai, so the squared error is $(ai - N)^2$.

If we play the game N times and see each train once, the mean squared error is

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(ai - N)^2$$

To minimize MSE, we take the derivative with respect to a:

$$\frac{dMSE}{da} = \frac{1}{N}\sum_{i=1}^{N}2i(ai - N) = 0$$

And solve for a.

$$a = \frac{3N}{2N+1}$$

At first glance, that doesn't seem very useful, because N appears on the right-hand side, which suggests that we need to know N to choose a, but if we knew N, we wouldn't need an estimator in the first place.

However, for large values of N, the optimal value for a converges to $3/2$, so we could choose $\hat{N} = 3i/2$.

To find an unbiased estimator, we can compute the mean error (ME):

$$ME = \frac{1}{N}\sum_{i=1}^{N}(ai - N)$$

And find the value of a that yields ME = 0, which turns out to be

$$a = \frac{2N}{N+1}$$

For large values of N, a converges to 2, so we could choose $\hat{N} = 2i$.

So far we have generated three estimators, i, $3i/2$, and $2i$, that have the properties of maximizing likelihood, minimizing squared error, and being unbiased.

Yet another way to generate an estimator is to choose the value that makes the population mean equal the sample mean. If we see train i, the sample mean is just i; the train population that has the same mean is $\hat{N} = 2i - 1$.

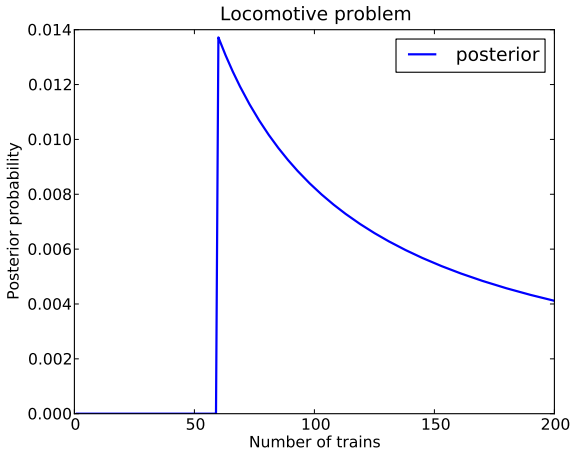

Figure 8.1: Posterior distribution of the number of trains.

Finally, to compute the Bayesian posterior distribution, we compute

$$P(H_n \mid i) = \frac{P(i \mid H_n)P(H_n)}{P(i)}$$

Where H_n is the hypothesis that there are n trains, and i is the evidence: we saw train i. Again, $P(i \mid H_n)$ is $1/n$ if $i < n$, and 0 otherwise. The normalizing constant, $P(i)$, is just the sum of the numerators for each hypothesis.

If the prior distribution is uniform from 1 to 200, we start with 200 hypotheses and compute the likelihood for each. You can download an implementation from `http://thinkstats.com/locomotive.py`. Figure 8.1 shows what the result looks like.

The 90% credible interval for this posterior is [63, 189], which is still quite wide. Seeing one train doesn't provide strong evidence for any of the hypotheses (although it does rule out the hypotheses with $n < i$).

If we start with a different prior, the posterior is significantly different, which helps to explain why the other estimators are so diverse.

One way to think of different estimators is that they are implicitly based on different priors. If there is enough evidence to swamp the priors, then all estimators tend to converge; otherwise, as in this case, there is no single estimator that has all of the properties we might want.

Exercise 8.6 Generalize `locomotive.py` to handle the case where you see more than one train. You should only have to change a few lines of code.

See if you can answer the other questions for the case where you see more than one train. You can find a discussion of the problem and several solutions at `http://wikipedia.org/wiki/German_tank_problem`.

8.10 Glossary

estimation: The process of inferring the parameters of a distribution from a sample.

estimator: A statistic used to estimate a parameter.

mean squared error: A measure of estimation error.

maximum likelihood estimator: An estimator that computes the point estimate with the highest likelihood.

bias: The tendency of an estimator to be above or below the actual value of the parameter, when averaged over repeated samples.

point estimate: An estimate expressed as a single value.

confidence interval: An estimate expressed as an interval with a given probability of containing the true value of the parameter.

credible interval: Another name for a Bayesian confidence interval.

censored data: A dataset sampled in a way that systematically excludes some data.