Contents

CHAPTER 1

Inferring information about models from samples

In the previous chapter, we were able to estimate the mean of a population from a sample. The only assumption we needed was that the sample was independent and random (the jar model); from this, we could get an estimate of the mean, and tell how good that estimate was.

In the previous chapter, we assumed that the population was a finite, but huge, dataset. In this chapter, we will instead model the population with a probability distribution. This model represents a potentially arbitrarily large supply of data. In the previous chapter, we obtained the sample by selecting a small subset of the data using a jar model. In this chapter, our probability model summarizes the population, and we assume that our data is produced by simulating that probability model. We must now take our data and determine *which* probability model applies. Generally, application logic suggests the type of model (i.e. normal probability density; Poisson probability; geometric probability; and so on). But usually, we do not know the parameters of the model — for example, the mean and standard deviation of a normal distribution; the intensity of a poisson distribution; and so on. Our model will be better or worse according as we choose the parameters well or badly. We need to be able to estimate the parameters of a model from a sample dataset.

1.1 DRAWING SAMPLES FROM A PROBABILITY DISTRIBUTION

We have already seen some methods to simulate random processes. We are now thinking of the data we have as the results of simulation of some probability model, but we don't know many simulation methods. While simulation doesn't directly tell us model parameters, it is useful to know some more about simulation.

Assume we have some way of producing data items x_i such that (a) they are independent; (b) each is produced from the same process; and (c) the histogram of a very large set of data items looks increasingly like the probability distribution $P(x)$ as the number of data items increases. Then we refer to these data items as **independent identically distributed samples** of $P(x)$; for short, **iid samples** or even just **samples**. The probability distribution here could be given by a probability density function, or a discrete probability distribution. If we are dealing with a probability density function, the histogram should have arbitrarily small boxes. In either case, the histogram should be normalized so that the sum of box areas is one.

This is a slightly technical way of capturing a natural intuition. A probability distribution can model an arbitrarily large dataset. Which particular model applies to which particular dataset is a matter of choice, judgement and quite often convenience (there's no point making a model one can't work with).

1.1.1 Simple Samples from Matlab

It is often useful to be able to draw samples from particular probability distributions. A great deal of ingenuity is spent on this point; I will just show a few tricks, on top of two standard random number generators. Matlab provides the function `rand`, which returns a number uniformly distributed in the range $[0-1]$. Furthermore, `rand(10, 20)` will get you a 10×20 table of independent uniformly distributed random numbers in the range $[0-1]$. Another really useful function is `randn`, which will give you a normal random variable, with mean zero and standard deviation one. Notice `randn(3, 4)` will give you a 3×4 table of such numbers, which are independent of each other. If you want a normal random variable with mean 3 and standard deviation 4, you use `4*randn+3`.

There are several other useful tricks. If you want a discrete random variable with uniform distribution, maximum value 100 and minimum value 7, I habitually choose a very tiny number (for this example, say 1e-7) and do `floor((100-7-1e-7)*rand+7)`. I do this because I can never remember whether `rand` produces a number no larger than one, or one that is guaranteed to be smaller than one, and I never need to care about the very tiny differences in probability caused by the 1e-7. You might be able to do something cleaner if you bothered checking this point.

As we have seen, if you want to shuffle a vector of 11 elements, you can use `randperm(11)` which will give a uniformly distributed random permutation of the numbers $1, 2, \ldots, 11$.

1.1.2 Rejection Sampling

Imagine you know a probability distribution describing a discrete random variable. In particular, you have one probability value for each of the numbers $1, 2, \ldots, N$ (all the others are zero). Write $p(i)$ for the probability of i. You can generate a sample of this distribution by the following procedure: first, generate a sample x of a uniform discrete random variable in the range $1, \ldots, N$; now generate a sample t of a uniformly distributed continuous random variable in the range $[0, 1]$; finally, if $t < p(x)$ report x, otherwise generate a new x and repeat. This process is known as **rejection sampling** (Algorithm 1.1).

Rejection sampling works because

$$\begin{aligned} P(\{\text{report } x \text{ in one round}\}) &= \left(\begin{array}{c} P(\{\text{accept } x\} \mid \{\text{generate } x\}) \\ \times \\ P(\{\text{generate } x\}) \end{array} \right) \\ &= p(x) \times \frac{1}{N}. \end{aligned}$$

The problem here is that if you don't report a sample in the first round (because t is too big), you have to generate another sample, and so on. This makes the current version of the algorithm absurdly inefficient. For example, if we want to generate a sample of a uniform distribution on the range $1, \ldots, N$, we may need many rounds (because $p(x)$ will be $1/N$, so t is often too big).

You can make things more efficient by noticing that multiplying the probability distribution by a constant doesn't change the relative frequencies with which

Listing 1.1: Matlab code for simple rejection sampling; this is inefficient, but simple to follow.

```
function rnum=rejectsample(pvec)
%
% pvec is a probability distribution over numbers
%  1, ..., size(pvec, 1)
%
nv=size(pvec, 1);
done=0;
while done==0
     ptr=floor(1+(nv-1e-10)*rand);
     % this gives me a uniform random
     % number in the range 1, .. nv
     pval=pvec(ptr);
     if rand<pval
          done=1;
          % i.e. accept
     end
end
rnum=ptr;
```

numbers are selected. So this means that we can find the largest value $\hat{p}$ of $p(x)$, and form $q(x) = (1/\hat{p})p(x)$. Our process then becomes that shown in algorithm 1.2.

This process is not the most efficient available. You could make it more efficient by building a binary tree whose leaves were the numbers $1, \ldots, N$, then flipping appropriately biased coins to walk the tree, reporting the leaf you arrive at (Algorithm 1.3 is the caller for the tree walk, Algorithm 1.4 does the work).

1.2 ESTIMATING PARAMETERS WITH MAXIMUM LIKELIHOOD

We have a coin, and would like to know $P(H) = p$ (where p is currently unknown). One strategy to identify p is to toss the coin k times, then look at the number of heads we see. This is (in essence) the strategy of the previous section. To see this, we assume that the coin is tossed an infinite number of times and we see a sample of k tosses, selected independently and at random. The strategy yields an estimate of p is

$$\frac{\text{number of heads}}{\text{number of tosses}},$$

which we know will be rather good if the number of tosses is large. But we could see this problem slightly differently. We know the probability distribution for the number of heads is a binomial distribution, and it has parameter p — we could try to wrestle information about p from our observed data.

We could apply another strategy to estimate p. We flip the coin repeatedly until we see a head. We know that, in this case, the number of flips has the geometric distribution with parameter p. Again, we could try to wrestle information about p

Listing 1.2: Matlab code for simple rejection sampling; this is somewhat more efficient.

```
function rnum=fasterrejectsample(pvec)
%
% pvec is a probability distribution over numbers
%  1, ..., size(pvec, 1)
%
nv=size(pvec, 1);
wv=max(pvec);
pv2=pvec/wv;  % we rescale
done=0;
while done==0
    ptr=floor(1+(nv-1e-10)*rand);
    % this gives me a uniform random
    % number in the range 1, .. nv
    pval=pv2(ptr); % work with rescaled probs
    if rand<pval
        done=1;
        % i.e. accept
    end
end
rnum=ptr;
```

Listing 1.3: Matlab code to call a sampler for a vector of probabilities. This is a wrapper that calls a recursive tree walk.

```
function ind=samplevec(vec)
ind=rsample(vec, 0);
```

from our observed data.

As yet another example, imagine we know for some reason that our data is well described by a normal distribution. We could ask what is the mean and standard deviation of the normal distribution that best represents the data?

The general problem is this: We have a set of observations D. We know that these observations can be described by a probability model that has some unknown parameters which are traditionally written θ. We must determine the parameter values.

Example: *Inferring p from repeated flips — binomial*

In this case, the data is a sequence of 1's and 0's from the coin flips. There are n flips (or terms) and k heads (or 1's). We know that an appropriate probability model is the binomial model $P(k; n, p)$. But we do not know p, which is the parameter.

Listing 1.4: This code walks a tree representation of the vector, flipping biased coins to come up with the sample.

```
function ind=rsample(vec, offset)
sv=max(size(vec));
if sv==1
  ind=offset+1;
elseif sv==2
  w1=sum(vec);
  r1=(w1*(1-1e-10))*rand+(1e-9*w1);
  if r1>vec(1)
    ind=offset+2;
  else
    ind=offset+1;
  end
else
  %
  % split it
  %
  nin=floor(sv/2);
  v1=vec(1:nin);
  v2=vec(nin+1:sv);
  tot=sum(vec);
  r1=(tot*(1-1e-10)*rand+(1e-9*tot));
  if r1<sum(v1)
    ind=rsample(v1, offset);
  else
    ind=rsample(v2, offset+nin);
  end
end
```

Example: *Inferring p from repeated flips — geometric*

In this case, the data is a sequence of 0's with a final 1 from the coin flips. There are n flips (or terms) and the last flip is a head (or 1). We know that an appropriate probability model is the geometric distribution $P_g(n; p)$. But we do not know p, which is the parameter.

1.2.1 The Maximum Likelihood Principle

In each of the examples above, we know the probability of observing the data, conditioned on a parameter value. We could write this as $P(D|\theta)$. We need a "reasonable" procedure to choose a value of θ to report. One — and we stress this is not the only one — is the **maximum likelihood principle**. This says: Choose θ such that $P(D|\theta)$ is maximised.

The maximum likelihood principle has a variety of neat properties we cannot yet expound. One worth knowing about is **consistency**; for our purposes, this means that the maximum likelihood estimate of parameters can be made arbitrarily close to the right answer by having a sufficiently large dataset.

Example: *Inferring p with maximum likelihood for repeated independent coin flips — binomial*

We see a sequence of 1's and 0's from independent coin flips. There are n flips (or terms) and k heads (or 1's). We know that an appropriate probability model is the binomial model $P(k; n, p)$. But we do not know p, which is the parameter.
We have that

$$P(D|\theta) = P_b(k; n, \theta)$$

which is a function of θ — the unknown probability that a coin comes up heads; k and n are known. We must find the value of θ that maximizes this expression. Now

$$P_b(k; n, \theta) = \left(\begin{array}{c} n \\ k \end{array}\right) \theta^k (1-\theta)^{(n-k)}$$

and the maximum occurs when

$$\frac{\partial P_b(k; n, \theta)}{\partial \theta} = 0.$$

So we can write

$$\frac{\partial P_b(k; n, \theta)}{\partial \theta} = \left(\begin{array}{c} n \\ k \end{array}\right) \left(k\theta^{k-1}(1-\theta)^{(n-k)} - \theta^k (n-k)(1-\theta)^{(n-k-1)}\right)$$

and this is zero when

$$k\theta^{k-1}(1-\theta)^{(n-k)} = \theta^k (n-k)(1-\theta)^{(n-k-1)}$$

so the maximum occurs when

$$k(1-\theta) = \theta(n-k)$$

or when

$$\theta = \frac{k}{n}$$

which is what we guessed would happen, but now we know why that guess "makes sense".

Example: *Inferring p with maximum likelihood for repeated independent coin flips — geometric*

We flip a coin n times, stopping when we see a head. We have a sequence of tails, ending in a head. There are n flips (or terms) and the last flip is a head (or 1). We know that an appropriate probability model is the geometric distribution $P_g(n; p)$. But we do not know p, which is the parameter.
We have that

$$P(D|\theta) = P_g(n;\theta) = (1-\theta)^{(n-1)}\theta$$

which is a function of θ — the unknown probability that a coin comes up heads; n is known. We must find the value of θ that maximizes this expression. Now the maximum occurs when

$$\frac{\partial P_g(n;\theta)}{\partial\theta} = 0.$$

So we can write

$$\frac{\partial P_g(n;\theta)}{\partial\theta} = ((1-\theta)^{(n-1)} - (n-1)(1-\theta)^{(n-2)}\theta)$$

and this is zero when

$$(1-\theta)^{(n-1)} = (n-1)(1-\theta)^{(n-2)}\theta$$

so the maximum occurs when

$$(1-\theta) = (n-1)\theta$$

or when

$$\theta = \frac{1}{n}.$$

We didn't guess this.

These two examples suggest some difficulties could occur in inference. It can be hard to find the maximum of the likelihood. Small amounts of data can present nasty problems — for example, in the binomial case, if we have only one flip we will estimate p as either 1 or 0. We cannot guarantee the right answer. In the geometric case, with a fair coin, there is a probability 0.5 that we will perform the estimate and then report that the coin has $p = 1$.

Example: *Inference with multinomial probabilities*

I throw a die n times, and see n_1 ones, … and n_6 sixes. Write $p_1, \ldots, p_6$ for the probabilities that the die comes up one, …, six. We now need to estimate $p_1, \ldots, p_6$. We could do this with maximum likelihood. The data are n, $n_1, \ldots, n_6$. The parameters are $\theta = (p_1, \ldots, p_6)$. $P(D|\theta)$ comes from the multinomial distribution. In particular,

$$P(D|\theta) = \frac{n!}{n_1! \ldots n_6!} p_1^{n_1} p_2^{n_2} \ldots p_6^{n_6}$$

which is a function of $\theta = (p_1, \ldots, p_6)$. Now we want to maximize this function by choice of θ. Notice that we could do this by simply making all p_i very large — but this omits a fact, which is that $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1$. So we substitute using $p_6 = 1 - p_1 - p_2 - p_3 - p_4 - p_5$ (there are other, neater, ways of dealing with this issue, but they take more background knowledge). At the maximum, we must have that for all i,

$$\frac{\partial P(D|\theta)}{\partial p_i} = 0$$

which means that, for p_i, we must have

$$n_i p_i^{(n_i-1)}(1 - p_1 - p_2 - p_3 - p_4 - p_5)^{n_6} - p_i^{n_i} n_6 (1 - p_1 - p_2 - p_3 - p_4 - p_5)^{(n_6-1)} = 0$$

so that, for each p_i, we have

$$n_i(1 - p_1 - p_2 - p_3 - p_4 - p_5) - n_6 p_i = 0$$

or

$$\frac{p_i}{1 - p_1 - p_2 - p_3 - p_4 - p_5} = \frac{n_i}{n_6}.$$

You can check that this equation is solved by

$$p_i = \frac{n_i}{n_1 + n_2 + n_3 + n_4 + n_5 + n_6}$$

Again, the maximum likelihood estimate is useful, and is consistent with intuition, but small datasets present some worries. If we throw the die only a few times, we could reasonably expect that, for some i, $n_i = 0$. This doesn't necessarily mean that $p_i = 0$ (though that's what this inference procedure will tell us). This creates a very important technical problem — how can I estimate the probability of events that haven't occurred? This might seem like a slightly silly question to you, but it isn't. For example, a really important part of natural language processing involves estimating the probability of groups of three words. These groups are usually known as "trigrams". People typically know an awful lot of words (tens to hundreds of thousands, depending on what you mean by a word). This means that there are a tremendous number of trigrams, and you can expect that any dataset lacks almost all of them. Some are missing because they don't occur in real life, but others are not there simply because they are unusual (eg "Atom Heart Mother" actually occurs in real life, but you may not have seen it all that often). Modern speech recognition systems need to know how probable every trigram is. Worse, if a trigram is modelled as having zero probability and actually occurs, the system will make a mistake, so it is important to model all such events as having a very small, but not zero, probability.

1.2.2 More Examples of Maximum Likelihood

It is often a lot easier to work with log-likelihood than with likelihood.

Worked example 1.1 *Poisson distributions*

You observe N intervals, each of the same, fixed length (in time, or space). You know that these events occur with a Poisson distribution (for example, you might be observing Prussian officers being kicked by horses...). You know also that the intensity of the Poisson distribution is the same for each observation. The number of events you observe in the i'th interval is n_i. What is the intensity, λ?

Solution: The likelihood is

$$\prod_i P(\{n_i \text{ events}\} \,|\lambda) = \prod_i \frac{\lambda^{n_i} e^{-\lambda}}{n_i!}$$

and it will be easier to work with logs. The log-likelihood is

$$\sum_i (n_i \log \lambda - \lambda - \log n_i!)$$

so that we must solve

$$\sum_i (\frac{n_i}{\lambda} - 1) = 0$$

which yields

$$\lambda = \frac{\sum_i n_i}{N}$$

Worked example 1.2 *Normal distributions*
Assume we have $x_1, \ldots, x_N$ which are data that can be modelled with a normal distribution. Use the maximum likelihood principle to estimate the mean of that normal distribution.

Solution: The likelihood of a set of data values under the normal distribution with mean μ and standard deviation σ is

$$\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

and this expression is a moderate nuisance to work with. But notice that the logarithm is a monotonically increasing function — this means that, if x gets bigger, so does $\log x$. We don't have to worry about negative numbers or zeros, because the normal distribution is everywhere positive. So if we were to maximize the log of the likelihood with respect to μ, that would give the same result as maximizing the likelihood wrt μ. The log of the likelihood is

$$\left(\sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2}\right) + \qquad \text{term not depending on } \mu.$$

We can find the maximum by differentiating wrt μ and setting to zero, which yields

$$\begin{aligned}\sum_{i=1}^{N} \frac{2(x_i-\mu)}{2\sigma^2} &= 0 \\ &= \frac{1}{\sigma^2}\left(\sum_{i=1}^{N} x_i - N\mu\right)\end{aligned}$$

so the maximum likelihood estimate is

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$$

which probably isn't all that surprising.

Worked example 1.3 *Normal distributions -II*
Assume we have $x_1, \ldots, x_N$ which are data that can be modelled with a normal distribution. Use the maximum likelihood principle to estimate the standard deviation of that normal distribution.

Solution: Now we have to write out the log of the likelihood in more detail; we get

$$\left(\sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2}\right) - N\log\sigma + \text{Term not depending on } \sigma$$

We can find the maximum by differentiating wrt σ and setting to zero, which yields

$$\frac{-2}{\sigma^3}\sum_{i=1}^{N}\frac{-(x_i-\mu)}{2} - \frac{N}{\sigma} = 0$$

so the maximum likelihood estimate is

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}$$

which probably isn't all that surprising, either.

Worked example 1.4 *Choosing Predictors with Maximum Likelihood*
Assume we wish to model a random variable Y as a linear function of another random variable X, plus noise. This noise will be a zero mean normal random variable, with unknown variance. Then our model says that $y - ax - b$ is a zero mean normal random variable. Equivalently, our model is that $P(y|x, a, b, \sigma)$ is a zero mean normal random variable with variance σ^2. We have a set of pairs (x_i, y_i) — what a and b should we choose? what will the value of σ be?

Solution: We want to maximize the likelihood

$$\prod_i P(y_i|x_i, a, b, \sigma) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y_i - ax_i - b)^2}{2\sigma^2}\right)$$

but this form looks inconvenient (among other things, we don't know σ). We take logs, to get

$$\mathcal{L}(a, b, \sigma) = \sum_i \log P(y_i|x_i, a, b, \sigma) = \sum_i \left[\left(\frac{-(y_i - ax_i - b)^2}{2\sigma^2}\right) - \log\sigma - \frac{1}{2}\log 2\pi\right].$$

To maximise this, we can take the partial derivatives and set them to zero. We have

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial a} &= \sum_i \frac{(y_i - ax_i - b)x_i}{\sigma^2}\\ \frac{\partial\mathcal{L}}{\partial b} &= \sum_i \frac{(y_i - ax_i - b)}{\sigma^2}\\ \frac{\partial\mathcal{L}}{\partial \sigma} &= \sum_i \left[\left(\frac{(y_i - ax_i - b)^2}{\sigma^3}\right) - \frac{1}{\sigma}\right]\end{aligned}$$

Now we can recover a, b and c by solving

$$\begin{pmatrix} \sum_i x_i^2 & \sum_i x_i \\ \sum_i x_i & N \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_i x_i y_i \\ \sum_i y_i \end{pmatrix}$$

and

$$\sum_i \left[(y_i - ax_i - b)^2 - \sigma^2\right] = 0$$

1.3 BAYESIAN INFERENCE

Sometimes when we wish to estimate parameters of a model we have prior information. For example, we may have good reason to believe that some parameter is close to some value. We would like to take this information into account when we estimate the model. One way to do so is to place a **prior probability distribution** $p(\theta)$ on the parameters θ. Then, rather than working with the likelihood $p(D|\theta)$, we could apply Bayes' rule, and form the **posterior** $p(\theta|D)$. This posterior represents the probability that θ takes various values, given the data D.

There are two natural ways to use the posterior. In the first, we plot the distribution, and try and infer useful information from it. In the second, sometimes known as **maximum a priori inference** or **MAP inference**, we choose the value of θ that maximizes the posterior.

Bayes' rule tells us that

$$p(D|\theta) = \frac{P(D,\theta)}{P(D)}$$

but (as we shall see) it can be hard to work out $P(D)$. However, $P(D)$ does not depend on θ — it depends only on the data. This means that we can often ignore it. If we wish to perform MAP inference, $P(D)$ doesn't matter (it changes the value, but not the location, of the maximum). If we wish to look at the posterior, $P(D)$ is just a scaling constant. So we usually write

$$p(D|\theta) \propto P(D,\theta)$$

and work with $P(D,\theta)$, often called the **joint** distribution.

Worked example 1.5 *Flipping a coin*

We have a coin with probability p of coming up heads when flipped. We start knowing nothing about p. We then flip the coin 10 times, and see 7 heads (and 3 tails). The model parameters are $\theta = p$. What is $p(p|\{7 \text{ heads and } 3 \text{ tails}\})$?

Solution: We know nothing about p, except that $0 \leq p \leq 1$. It is reasonable then that the prior on p is uniform. We have that $p(\{7 \text{ heads and } 3 \text{ tails}\}|p)$ is binomial. It is enough to work with the joint, which is

$$p(\{7 \text{ heads and } 3 \text{ tails}\}|p) \times p(p)$$

but $p(p)$ is uniform, so doesn't depend on p. So the posterior is *proportional* to:

$$\binom{10}{7} p^7(1-p)^3$$

which is graphed in figure 1.1.

Worked example 1.6 *Flipping a coin - II*

We have a coin with probability p of coming up heads when flipped. We start knowing nothing about p. We then flip the coin 10 times, and see 7 heads (and 3 tails). The model parameters are $\theta = p$. What is $P(\{\theta \in [0.5, 0.8]\})$?

Solution: We could answer this question by computing

$$\int_{0.5}^{0.8} p(\theta|D) d\theta$$

We have already worked out that the posterior is *proportional* to

$$\theta^7 (1-\theta)^3.$$

Now we could work out the constant of proportionality k by noticing that $p(\theta|D) = k\theta^7(1-\theta)^3$, and

$$\int_0^1 p(\theta|D) d\theta = 1.$$

This means

$$k = \frac{1}{\int_0^1 \theta^7 (1-\theta)^3 d\theta}$$

We are looking for a number, so that a numerical estimate of the constant of proportionality is fine. I obtained one by numerical integration; I divided the interval of θ values (which is $0-1$) into 1000 steps, evaluated the function at each point, then estimated the integral as

$$(\sum_i \theta^7 (1-\theta)^3)(1/1000)$$

although you might know cleaner methods. This got me the value $k = 1.32e3$. Now we can compute an estimate of $\int_{0.5}^{0.8} p(\theta|D) d\theta$ using a similar approach; I got

$$\int_{0.5}^{0.8} p(\theta|D) d\theta \approx 0.7238$$

Example **??** can be generalized. Again, assume that the prior probability distribution is uniform. In this case the posterior is proportional to the likelihood (check this statement), so MAP inference isn't all that interesting. But it is interesting to follow how our posterior on p changes as evidence comes in, which is easy to do because the posterior is proportional to a binomial distribution. Figure 1.2 shows a set of these posteriors for different sets of evidence.

1.3.1 Normal Distributions and Bayesian Inference

Usually, we don't have very good reasons to choose one prior model over another. However, there are some prior properties that we could try to impose. One is that

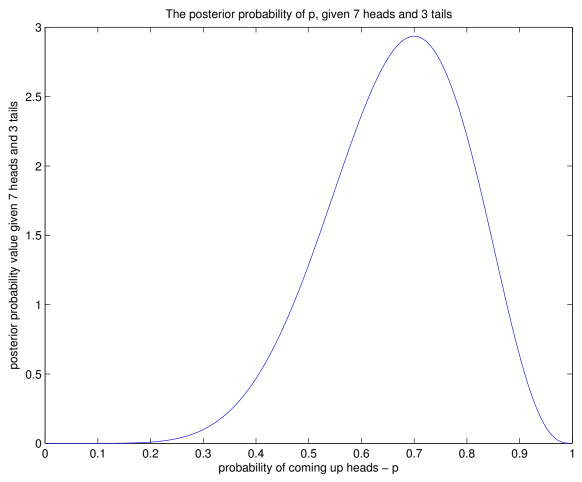

FIGURE 1.1: *The probability that an unknown coin will come up heads when flipped is p. We assume the coin could have any p — perhaps it comes from someone known for manipulating coins — and so the prior on p is uniform. We now flip the coin, and get 7 heads and 3 tails. The curve shows the posterior on p in this case, which you can think of as the probability that p will take a particular value, given this evidence. Notice that this information is rather richer than the single value we would get from maximum likelihood inference. The MAP value of p is 0.7, but this posterior shows that p could still take quite a range of numbers.*

unknown parameters tend to be small, or tend to be close to some value. Normal distributions are particularly useful in this case.

We start with a simple example. Assume we drop a measuring device down a borehole. It is designed to stop falling and catch onto the side of the hole after it has fallen μ_0 meters. On board is a device to measure its depth. This device reports a known constant c times the correct depth plus a zero mean normal random variable, which we call "noise". This noise has standard deviation σ_n. The device reports depth every second.

The first question to ask is what depth do we believe the device is at *before* we receive any measurement? We designed the device to stop at μ_0 meters, so we are not completely ignorant about where it is. However, it may not have worked absolutely correctly. We choose to model the depth at which it stops as μ_0 meters plus a zero mean normal random variable. The second term could be caused by error in the braking system, etc. We could estimate the standard deviation of the second term (which we write σ_0) either by dropping devices down holes, then measuring with tape measures, or by analysis of likely errors in our braking system. The depth of the object is the unknown parameter of the model; we write this depth d. Now the model says that $P(d)$ is a normal random variable with mean μ_0 and standard deviation σ_0.

Notice that this model probably isn't exactly right — for example, there must be some probability in the model that the object falls beyond the bottom of the

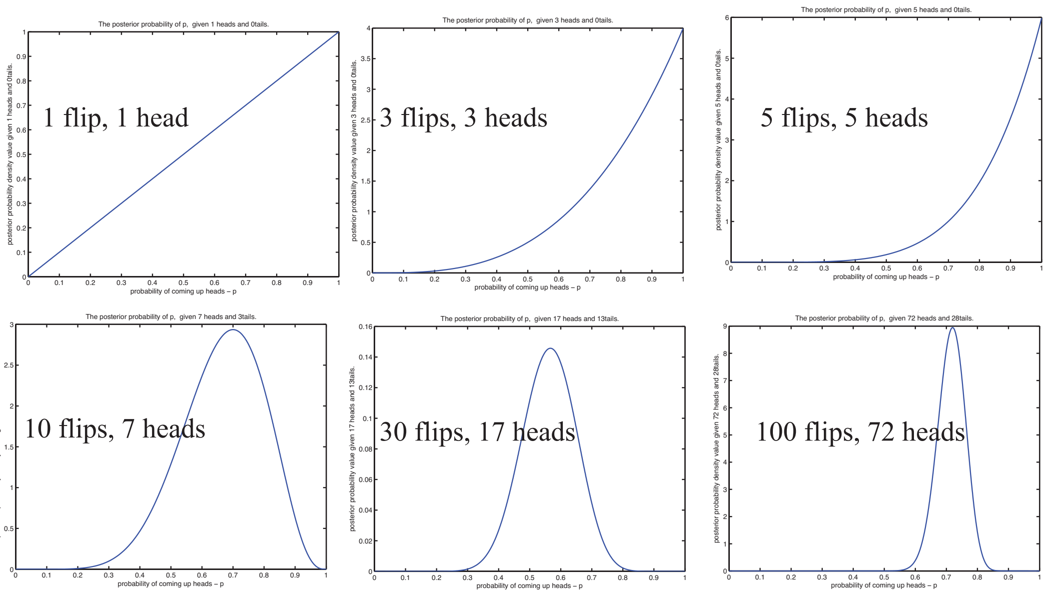

FIGURE 1.2: *The probability that an unknown coin will come up heads when flipped is p. For these figures, I simulated coin flips from a coin with $p = 0.75$. I then plotted the posterior for (***top row***, in order) 1 flip, 3 flips, 5 flips, (***bottom row***, in order) 10 flips, 30 flips and 100 flips. In this case, the coin came up heads five times before the first tail. Notice how the posterior gets bigger at the higher values of p, but is not tightly peaked. As we see more flips, we get more confident about p. One attraction of Bayesian inference is we can keep track of our uncertainty on the parameter we are estimating, by looking at the posterior.*

hole, which it can't do — but it captures some important properties of our system. The device should stop at or close to μ_0 meters most of the time, and it's unlikely to be too far away.

Now assume we receive a single measurement — what do we now know about the device's depth? The first thing to notice is that there is something to do here. Ignoring the prior and taking the measurement might not be wise. For example, imagine that the noise in the wireless system is large, so that the measurement is often corrupted — our original guess about the device's location might be better than the measurement. Write x for the measurement. Notice that the scale of the measurement may not be the same as the scale of the depth, so the mean of the measurement is cd, where c is a change of scale (for example, from inches to meters). We have that $p(x|d)$ is a normal random variable with mean cd and standard deviation σ_n. We would like to know $p(d|x)$.

Notice $p(d|x) \propto p(d,x)$ and $p(d,x) = p(x|d)p(d)$. In turn, this means that $\log p(d,x) = \log p(x|d) + \log p(d)$. We can write this out as

$$\log p(x|d) + \log p(d) = -\frac{(x-cd)^2}{2\sigma_n^2} - \frac{(d-\mu_0)^2}{2\sigma_0^2} + \text{terms not depending on } d \text{ or } x.$$

We have two estimates of the position, d, and we wish to come up with a representation of what we know about d. One is x, which is a *measurement* — we know its value. The expected value of x is cd, so we could infer d from x. But we have another estimate of the position, which is μ_0. The posterior, $p(d|x)$, is a probability distribution on the variable d; it depends on the known values x, μ_0, σ_0 and σ_n. We need to determine its form. We can do so by some rearrangement of the expression for $\log p(d,x)$. Notice first that this expression is of degree 2 in d (i.e. it has terms d^2, d and things that don't depend on d). The terms that don't depend on d are expressions in x, etc., that are constant for particular instances of the problem. This means that the probability distribution must be normal, because we can rearrange its log into the form

$$-\frac{(d-\mu_p)^2}{2\sigma_p^2} + \text{terms not depending on } d.$$

The terms not depending on d are not interesting, because if we know σ_p those terms must add up to

$$+\log\left(\frac{1}{\sqrt{2\pi}\sigma_p}\right)$$

so that the probability density function sums to one. Our goal is to rearrange terms into the form above. Notice that

$$-\frac{(d-\mu_p)^2}{2\sigma_p^2} = -d^2\left(\frac{1}{2\sigma_p^2}\right) + 2d\frac{\mu_p}{2\sigma_p^2} + \text{term not depending on } d$$

We have

$$\begin{aligned}\log p(d|x) &= -\frac{(cd-x)^2}{2\sigma_n^2} - \frac{(d-\mu_0)^2}{2\sigma_0^2} + \text{terms not depending on } d\\ &= -d^2\left(\frac{1}{2\left(\frac{\sigma_n^2\sigma_0^2}{\sigma_n^2+c^2\sigma_0^2}\right)}\right) + 2d\left(c\frac{x}{2\sigma_n^2} + \frac{\mu_0}{2\sigma_0^2}\right) + \text{terms not depending on } d\end{aligned}$$

which means that

$$\sigma_p^2 = \frac{\sigma_n^2\sigma_0^2}{\sigma_n^2 + c^2\sigma_0^2}$$

and

$$\begin{aligned}\mu_p &= 2\left(c\frac{x}{2\sigma_n^2} + \frac{\mu_0}{2\sigma_0^2}\right)\frac{\sigma_n^2\sigma_0^2}{\sigma_n^2+c^2\sigma_0^2}\\ &= \left(\frac{cx\sigma_0^2 + \mu_0\sigma_n^2}{\sigma_n^2\sigma_0^2}\right)\frac{\sigma_n^2\sigma_0^2}{\sigma_n^2+c^2\sigma_0^2}\\ &= \frac{cx\sigma_0^2+\mu_0\sigma_n^2}{\sigma_n^2+c^2\sigma_0^2}.\end{aligned}$$

The first thing to notice about these equations is that they "make sense". Imagine that σ_0 is very small, and σ_n is very big; then our new expected value of d

— which is μ_p — is about μ_0. Equivalently, because our prior was very accurate, and the measurement was unreliable, our expected value is about the prior value. Similarly, if the measurement is reliable (i.e. σ_n is small) and the prior has high variance (i.e. σ_0 is large), then our expected value of d is about x/c — i.e. the measurement, rescaled.

There are two ways to look at $p(d|x)$. One way is to think of it as a normal distribution with mean μ_p and variance σ_p^2. Another is to say we have an estimate of d — which is μ_p — and we know the standard deviation of that estimate (σ_p).

1.3.2 Repeated Bayesian Inference, or Filtering

In the last example, we had a prior distribution on d. This was a normal distribution with mean μ_0 and standard deviation σ_0. We then received a measurement x, where $p(x|d)$ was normal with mean cd and standard deviation σ_n. We were then able to compute a posterior on d, whose mean was μ_p and whose standard deviation was σ_p.

Now imagine we receive another measurement. We can apply our reasoning to obtain a new posterior on d. This is because our reasoning showed how to compute a posterior from a normal prior with a normal measurement. But the posterior is normal, so we can go again.

We can do more; imagine the measurement device, for reasons of its own, changes scale at each measurement, *and we know what the scales are.* For example, it could report depth in meters, then in centimeters, then in inches, then in feet, and so on. Because our analysis applies to a single measurement, we can cope.

The procedure looks like this. At the i'th step, before we receive a measurement, we have a prior distribution on d. This is normal, with mean μ_i and standard deviation σ_i. We now receive a measurement, which is normal, with mean $c_i d$ and standard deviation σ_n. By the equations above, the posterior is normal, with mean

$$\mu_p = \frac{c_i x \sigma_i^2 + \mu_i \sigma_n^2}{\sigma_n^2 + c_i^2 \sigma_i^2}$$

and variance

$$\sigma_p^2 = \frac{\sigma_n^2 \sigma_i^2}{\sigma_n^2 + c_i^2 \sigma_i^2}.$$

Now we can treat this posterior as the prior for the next measurement, so we have

$$\begin{aligned} \mu_{i+1} &= \frac{c_i x \sigma_i^2 + \mu_i \sigma_n^2}{\sigma_n^2 + c_i^2 \sigma_i^2} \\ \sigma_{i+1}^2 &= \sigma_p^2 = \frac{\sigma_n^2 \sigma_i^2}{\sigma_n^2 + c_i^2 \sigma_i^2}. \end{aligned}$$