Lab 5: Bayesian analysis of a dice toss problem

Due date: Friday March 9, 11:59pm

Short version of the assignment

Each of you get a 4D die: Figure out whether the die is fair or unfair, report most likely probabilities for each side. Write a report and send your findings (including the code and the data) to Kyle.

Slightly longer version of the assignment

Use the die I gave you to throw, say 50 times, and record the outcome (it will be 1,2,3, or 4), this will be your data D for the exercise. We will construct a program that is doing Bayesian inference and estimate the posterior probabilities $P(p|D)$ of the probability p of each side of the die. A short refresher on Bayesian inference: Bayes theorem suggests that we can get probability of the parameters of a model (your p) given the data D but assuming some distribution of the parameters and also knowing how to calculate the likelihood that the data fits a particular model (with a specified set of p values), then we can formulate:

$$P(p|D) = \frac{P(p)P(D|p)}{P(D)}, \tag{1}$$

the quantity in the denominator is a scaler so that the posterior distribution integrates to 1.0, thus we could say the $P(D)$ is the integral over all possible values of p: $\int_p P(p)P(D|p)dp$, but for our analysis we can dodge the calculation of this because we use Markov chain Monte Carlo to estimate our quantities. We thus can use

$$P(p|D) \propto P(p)P(D|p). \tag{2}$$

Our task can be broken down into 4 steps:

1. Construct the likelihood function
2. Construct the prior
3. Construct Markov chain Monte Carlo sampler (including a method how to change the p using our prior)
4. Visualize the results, print means etc.

1 Likelihood

We observe results that could be summarized like this: 1: 5, 2: 10, 3:6, 4:9. We have 5 throws that resulted in a 1, 10 throws for 2, 6 throws for 3 and 9 throws for 4, for a total of 30 throws. We will use this data further as a list [5,10,6,9], or more abstract [a,b,c,d]. If we would use a coin that we could report heads or tails and would use a binomial distribution, but for our problem we have 4 sides, thus will need an extension of the binomial and use the *multinomial distribution*, that can be calculated like this

$$P(D|p) = \frac{n!}{a!b!c!d!} p_1^a p_2^b p_3^c p_4^d = \frac{30!}{5!10!6!9!} p_1^5 p_2^{10} p_3^6 p_4^9 \quad (3)$$

The problem with this is that the result will be difficult for large numbers of throws, for example 100 or 200 throws will result in problems to calculate the factorials, a remedy to this is to operate all calculations in logs, if we do that then we get

$$\log P(D|p) = (\mathrm{logf}(n) - (\mathrm{logf}(a) + \mathrm{logf}(b) + \mathrm{logf}(c) + \mathrm{logf}(d))) + a\log p_1 + b\log p_2 + c\log p_3 + d\log p_4 \quad (4)$$

We could calculate $logf$ as the log of a factorial but that breaks with large numbers, we approximate using this

$$\log(x!) \equiv gammaln(x+1) \quad (5)$$

gammaln is available within numpy or scipy.

2 Prior

we will use prior that can take the p and calculate probability density function, appropriate for our problem is the Dirichlet distribution that takes p assuming that the p sum to 1 and also uses a set of parameters, we are laze and use a vector α with all ones, for our problem $\alpha = [1, 1, 1, 1]$, this is equivalent to flat prior where we believe all p come from the same distribution. The Dirichlet PDF can be coded using the sample code in this post
`http://stackoverflow.com/questions/10658866/calculating-pdf-of-dirichlet-distribution-in-python`,
take the code and create a function that may look like this:
`def pdf_dirichlet(p, alpha): .....` The pdf_dirichlet() function delivers a probability and takes two arguments, a vector p (these are the posterior values of the throw-probabilities and α these are the weights that lead to p. Recent Scipy versions contain the PDF of the Dirichlet and you can use *scipy.stats.dirichlet.pdf*(x, α) with a vector x and the concentration parameter α.

3 Markov chain Monte Carlo (MCMC)

- Propose new values p: We can propose new values for p from the prior, this is easy because numpy HAS a function for that: `np.random.dirichlet(alpha)` where `alpha` is our α from above.
- Start with an arbitrary value for example `p=np.random.dirichlet(alpha)`, evaluate the posterior with these p, `post=pdf_dirichlet(p, alpha) * like(data,p)`, then run for a large number of cycles through this recipe:

1. propose new p
2. evaluate the new posterior `new`
3. compare `new` with `old` (see above the `post` that probably should better called `old`); if the `new` is better than the `old` we will accept the new p and record it (for example append it to results), if `new` is smaller than `old` we accept with some probability r, this can be done easily using a condition $r < new/old$, but remember, we used logs to calculate all quantities, so our condition turns into this:

```
r = numpy.random.uniform(0,1)
if np.log(r) < new - old:
    append new p to results
    oldp = p
    old = new
else:
    append old p to results
```

The results contain now a chain of 'accepted' p values, a histogram of these will represent the posterior.

4 Visualize results

Use a histogram to show the bars for each posterior for each side of the die. Check out the `hist` examples. Discuss your results, if you are adventurous, try to calculate the credibility intervals.

Peter Beerli February 2017