Dublin R Workshop on Bayesian Data Analysis

Mick Cooney
michael.cooney@applied.ai

March 2016

`https://bitbucket.org/kaybenleroll/dublin_r_workshops`.

Code is available in the `wsbda201603/` directory.

Most of content of this workshop is based on "Doing Bayesian Data Analysis" by John Kruschke.

`https://sites.google.com/site/doingbayesiandataanalysis/`.

Another excellent book is "Data Analysis Using Regression and Multilevel/Hierarchical Models" by Gelman and Hill

`http://www.stat.columbia.edu/~gelman/arm/`.

Another excellent book is "Bayesian Data Analysis" by Gelman, Carlin, Stern, Dunson, Vehtari, and Rubin

`http://www.stat.columbia.edu/~gelman/book/`.

Finally, I highly recommend the book "Statistical Rethinking" by Richard McElreath. It teaches basic statistical principles from a Bayesian perspective.

`http://xcelab.net/rm/statistical-rethinking/`

1. Conditional Probability

Suppose that in the general population, the probability of having a specific rare disease (the Dreaded Lurgy) is one in a thousand. We denote the true presence or absence of the disease as the value of a parameter, θ, that can have the value 1 if disease is present, or the value 0 if the disease is absent. The base rate of the disease is therefore denoted $p(\theta = 1) = 0.001$. This is our prior belief that a person selected at random has the disease.

Suppose that there is a test for the disease that has a 99% hit rate, which means that if a person has the disease, then the test result is positive 99% of the time. We denote a positive test result as $D = 1$, and a negative test result as $D = 0$. The observed test result is a bit of data that we will use to modify our belief about the value of the underlying disease parameter. The hit rate is expressed as $p(D = 1 \mid \theta = 1) = 0.99$.

The test also has a false alarm rate of 5%. This means that 5% of the time when the disease is not present, the test falsely indicates that the disease is present. Wedenote the false alarm rate as $p(D = 1 \mid \theta = 0) = 0.05$.

However, what we need to know is $p(\theta = 1 \mid D = 1)$, i.e. the probability that the patient has the disease given a positive test result.

We can calculate the above conditional probability given Bayes' Rule and using arithmetic and algebra. Somewhat counter-intuitively, we calculate a probability slightly below 2%.

We will try to estimate this probability using simulation.

Exercise 1.1 The supplied function `generate.disease.test.data()` generates sample data based on the above numbers. Look at the function definition to see how it is used, and generate some sample data, then use this data to estimate the conditional probability. with a reasonable number of datapoints (say 1,000,000) you should get an estimate close to the analytic answer.

Exercise 1.2 Why is this conditional probability so low? Experiment with the dependency of this probability on the three input probabilities. (Any plotting system will work, though I will use ggplot2 for mine)

Exercise 1.3 How are the probabilities changed if two independant tests are tried? The provided function `generate.disease.twotest.data()` generates random data for this. Using this function, calculate the conditional probabilities of having the disease, given the various combinations of test results.

2. Analytical Approach

Bayesian reasoning is as old as the concept of probabilities, but has only recently started to receive a lot of attention. One likely reason for this is that, apart from a few special cases, it is mot possible to perform the calculations analytically.

In our application we have a prior distribution for our beliefs, $p(\theta)$, and a likelihood for the data, $p(D|\theta)$, and through use of the Chain Rule, we get the posterior distribution, $p(\theta|D)$,

$$p(\theta|D) = \int d\theta \, p(D|\theta) \, p(\theta).$$

In those special cases, the likelihood function has a prior and posterior distribution with the same functional form, i.e. the prior and the posterior are two members of the same 'family' of functions.

For the rest of this workshop we are going to deal with estimating the fairness of a coin, based on the result of multiple coin tosses. We define 'success' as the toss coming up Heads, and denote this probability as θ. Thus,

$$P(y = 1|\,\theta) = \theta \quad \text{and} \quad P(y = 0|\,\theta) = 1 - \theta.$$

We can combine the above two into a single expression:

$$P(y|\theta) = \theta^y (1 - \theta)^{(1-y)}$$

.

Now we consider the data y to be fixed, and consider the above as a function of θ. With this approach we call the above equation *the likelihood function of* θ.

The Bernoulli function has a conjugate prior: the *Beta distribution*, Beta(a, b). R supports the beta distribution natively, via the standard grouping of functions for probability distributions: `rbeta()`, `dbeta()`, `pbeta()`, `qbeta()`.

Exercise 2.1 Plot the density distribution for the Beta distribution using various combinations of parameters a and b.

Exercise 2.2 Load the data in `cointoss10`, and use it and the beta distribution to investigate the effect of the prior on the posterior distribution.

Exercise 2.3 Load the data in `cointoss1000`, and use it and the same priors used in the previous exercise to investigate the effect of the prior on the posterior distribution.

Exercise 2.4 Compare the posterior distributions for the same priors using the two datasets. How do they compare, and why do you think this is?

3. Numerical Solutions using a Discrete Grid

For many applications, the use of simple conjugate priors is not appropriate, and we need to deal with the posterior integral calculation itself. Since analytical solutions do not exist, we use numerical techniques to approximate the integral.

The supplied functions `calculate.data.probability()` and `calculate.posterior.probability()` perform these calculations. The major benefit of this approach is that we can now use arbitrary priors.

Exercise 3.1 Using the `cointoss10` data, and a Beta$(1, 1)$ prior, use the grid approximation to the integral to calculate the posterior density. Compare the output of this to the analytical solution.

Exercise 3.2 Repeat the previous exercise of comparing the influence of priors on the posterior density for both the 10 coin toss and 1,000 coin toss data. Match this output to the analytics solutions you derived earlier.

Exercise 3.3 Suppose we think the coin has a 3/1 bias, but we do not know for which side. Create a prior that represents this and investigate the posterior for both sets of data.

Exercise 3.4 Estimate the posterior density for the bias in the coin assuming you have an equally weighted prior belief of the coin being fair, or biased 3/1 for either side.

Exercise 3.5 Investigate the influence of the size of the dataset on the posterior for arbitrary priors.

4. Introducing Hierarchical Models

We can now extend this idea by using a hierarchical model. Previously, we were considering the coin by itself, and the goal was to estimate the parameters of the model. We could extend that model by using multiple coins, but our initial intuition would be to treat multiple coins as being independent of one another.

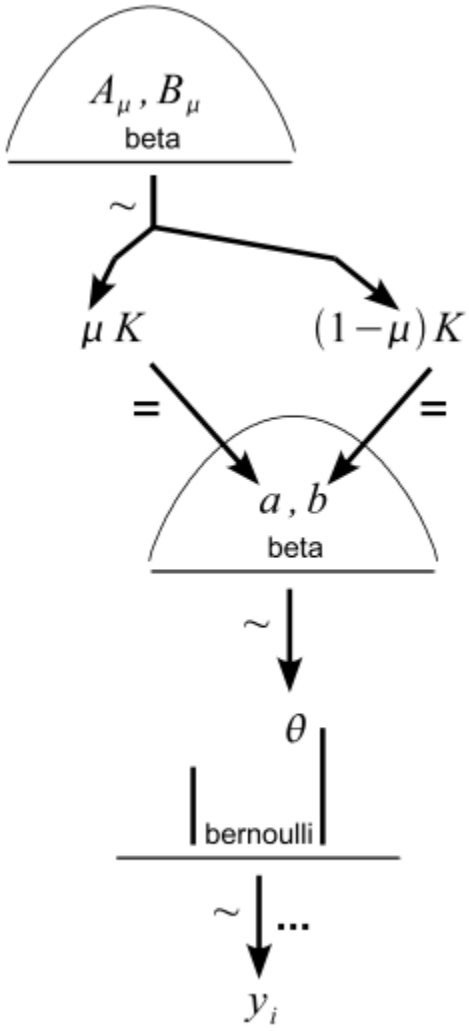


Figure 1: Graphical Representation of the Single Mint, Single Coin Hierarchical model

However, what if we were instead to also consider the fact that a coin is minted, and the bias of the coin is probably influenced by the manufacturing processes and quality of its origin. In this case, we could incorporate prior beliefs about the mint as well the coin, and then use our data to update those beliefs.

To start with, we will work from a single coin coming from a single mint. Each coin is produced with a bias of θ, but these are also random variables drawn from another Beta distribution with mean μ. This means that outcomes of the coin tosses, y, are the result of a hierarchical random process. At the top, we have a distribution for μ, and we call this the *hyperdistribution*. Accordingly, μ is a *hyperparameter*. We then use

the values of the hyperparameters as inputs to the distribution for *theta*. In turn, with the values of θ, we then determine values for the data, y. A graphical representation of this model is shown in Figure 1.

So, the hyperparameter μ is used to determine values A_μ, and B_μ for the hyper-beta-distribution. To do this, we also need a value K, which for the moment we will assume is a constant, and is a representation of how closely the value of θ depends on μ. For high values of K, θ will cluster tightly around μ.

Expressing this formally, we have

$$p(y|\theta) = \theta^y(1-\theta)^{(1-y)},$$

just like before, but we now also have θ as a random variate,

$$p(\theta|\mu) = B(\theta \,|\, \mu K,\, (1-\mu)K)$$

with μ itself being drawn from a hyperdistribution (in this case, the beta distribution):

$$\mu \backsim B(A_\mu, B_\mu).$$

Note that this model will required values for A_μ and B_μ, and these will reflect our prior beliefs about the mint. For this, we will probably just use low constant values, representing weak prior beliefs on the mint itself.

So, the above seems like a very interesting approach, but how do we go about implementing the approach? It is unlikely that an analytic solution would be either tractable or practical, but like the drunk looking for his keys, this is where the light is.

So, we know from Bayes' Rule that

$$p(\theta, \mu|y) = \frac{p(y|\theta,\mu)p(\theta,\mu)}{p(y)}.$$

From our model, we can see that, conditional on θ, the likelihood model does not depend on μ, and so we have

$$p(y|\,\theta, \mu) = p(y|\theta).$$

Also, from another application of Bayes' Rule,

$$p(\theta|\mu) = \frac{p(\theta,\mu)}{p(\mu)} \implies p(\theta,\mu) = p(\theta|\mu)\,p(\mu)$$

,
resulting in the following expression for the posterior distribution

$$p(\mu, \theta \,|\, y) = \frac{p(y|\theta)\,p(\theta|\mu)\,p(\mu)}{p(y)}$$

So, if we create a grid for both θ and μ, we can approximate the posterior from numerical integration. We take our models for $p(y|\theta)$, $p(\theta|\mu)$ and $p(\mu)$, and create our posterior by integration.

Exercise 4.1 Use a very weak prior for μ, but a value of 5 for K, and generate the posterior density for this model using the `cointoss10` data. You can use the supplied function `calculate.hierarchical.posterior()` to do this. How would you visualise this output in a meaningful way?

Exercise 4.2 How does the data affect the posterior distribution for μ and θ?

Exercise 4.3 Repeat the above exercise but with a K value of 100. Do you notice a difference in the posterior distributions of μ and θ?

Exercise 4.4 Repeat the above for $K = 1,000$. How are the posterior densities affected now?

Exercise 4.5 Repeat all of the above using the `cointoss1000` data.

Exercise 4.6 What impact does the size of the dataset have on the various posterior densities?

5. Markov-Chain Monte Carlo (MCMC) for Hierarchical Models

It should be quickly apparent that using the discrete grid approach to approximating these posterior densities will not be feasible computationally once we start adding more than two or three parameters over anything more than 100 to 1000 discrete steps in each dimension. The grids become very large very quickly, and even the incredible improvements in computational power have not been able to keep up.

Instead, we use a technique known as Markov-Chain Monte Carlo to sample from the posterior distribution. For this we use Stan, a generic MCMC sampler that uses Hamiltonian Monte Carlo, which has benefits over Gibbs sampling. Stan has largely superseded the older BUGS and JAGS software in the last few years. To use Stan with R, we need access to a C++ compiler and the package `rstan`.

There are a few steps involved in setting up the sampler, the first of which involves specifying the model used by Stan. Stan has its own language for doing this, and is often written into its own file. For our first model, the single-mint-single-coin model is provided in file `singlemint_singlecoin.stan`.

Exercise 5.1 Using the data in `cointoss10.rds`, set up the model in `rstan` and run the samples. What inferences can be made on μ from this model? Think about why this is the case.

Exercise 5.2 Repeat all the exercises from the equivalent grid approximation of this model, this time using `rstan` code to produce your distributions.

Exercise 5.3 Compare the prior and posterior distributions for θ.

Exercise 5.4 The above exercises involve changing some of the prior values. Could we redo the model to make it more efficient to change these priors?

Exercise 5.5 Discuss the validity of treating this scenario as a hierarchical model. What does μ represent in this case?

We now have all the tools we need to build hierarchical models, so let us extend the above scenario in more interesting ways. What if we have multiple coins minted from the same mint? What types of inferential models can be built from this set up?

The Stan model for a single mint, multiple coin model is supplied in the file `singlemint_multiplecoin.stan`. Open the file and make sure you understand the model. Data for multiple coin tosses with two different coins is given in the file `singlemint_twocoin.rds`. This is a data.table with three columns denoting the id of the coin, the number of trials and the number of successes.

Exercise 5.6 Create a prior and posterior sample for the above scenario. What kind of methods can we use to visualise this data? Note that these hierarchical models quickly prove problematic in terms of visualisation, so this is a real issue.

Exercise 5.7 Investigate how different values for K change the inferences. Check the differences in how the data influences the posterior distribution for any given prior.

Exercise 5.8 Think about how you would extend this model to tosses of three coins, and then five coins? How feasible is it to analyse data from the tosses of twenty coins?

6. Expanding Hierarchical Models

The true power of the Bayesian approach is that you can encapsulate any and all uncertainty in your modelling via your priors. Previously, we did have one parameter that we were 'guessing' at the value of, K, which controls the strength of the dependency of the coin mint bias μ on the bias of any individual coin, θ. For higher values of K, the values of θ are more closely clustered around the μ value for the mint.

To illustrate the influence of this value, we were running our sampler with different values of K. However, in reality, we give this value a prior distribution and allow the data to perform some inference on this value as well. The distribution we use for this one is the *Gamma distribution*, which is a common distribution for values $x \geq 0$.

This distribution is closely related to the *Gamma function*, $\Gamma(s) = \int_0^\infty dt\, t^{s-1}e^{-t}$, a generalisation for factorials.

The Gamma distribution has two parameters controlling it, the *rate* and the *shape*, and we set both these values so that our prior is broad across the possible values for κ.

The Gamma distribution is the conjugate prior for a Poisson distribution, and is analagous to the role that the Beta distribution holds for a binomial likelihood. An intuitive way of thinking about a particular Gamma distribution is to think of it as the distribution of the Poisson count rate when we observe *rate* events in *shape* units of time. Thus, the ratio of the two is the mean of the distribution,

$$\lambda = \frac{rate}{shape}$$

and the spread of the distribution is determined by the sizes of the *shape* or the *rate*.

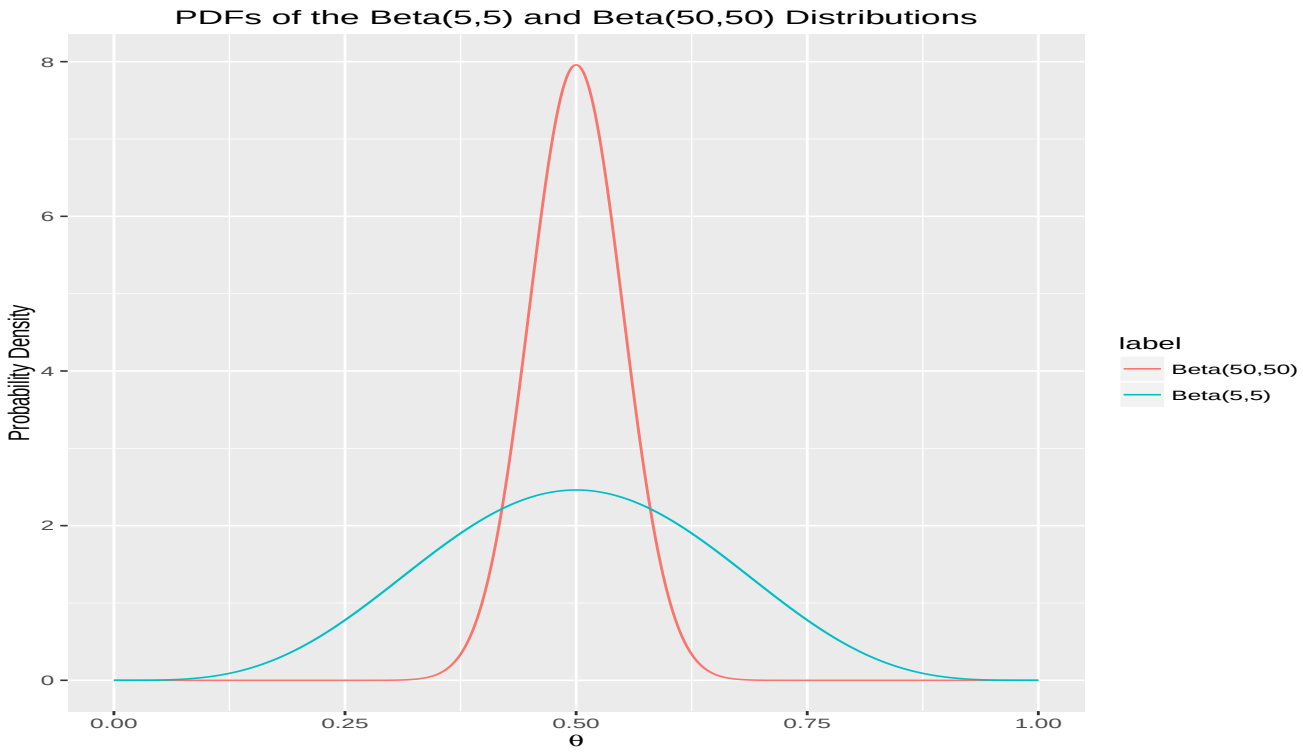
As always, graphs make this much clearer, so we show comparisons for the Beta and Gamma distributions

```
theta_seq <- seq(0, 1, by = 0.001)

beta_05_05 <- dbeta(theta_seq,  5,  5)
beta_50_50 <- dbeta(theta_seq, 50, 50)

plot_dt <- rbind(data.table(label   = 'Beta(5,5)'
                            ,theta   = theta_seq
                            ,density = beta_05_05)
                 ,data.table(label   = 'Beta(50,50)'
                            ,theta   = theta_seq
                            ,density = beta_50_50))

ggplot(plot_dt) +
    geom_line(aes(x = theta, y = density, colour = label)) +
    xlab(expression(theta)) +
    ylab("Probability Density") +
    ggtitle("PDFs of the Beta(5,5) and Beta(50,50) Distributions")
```



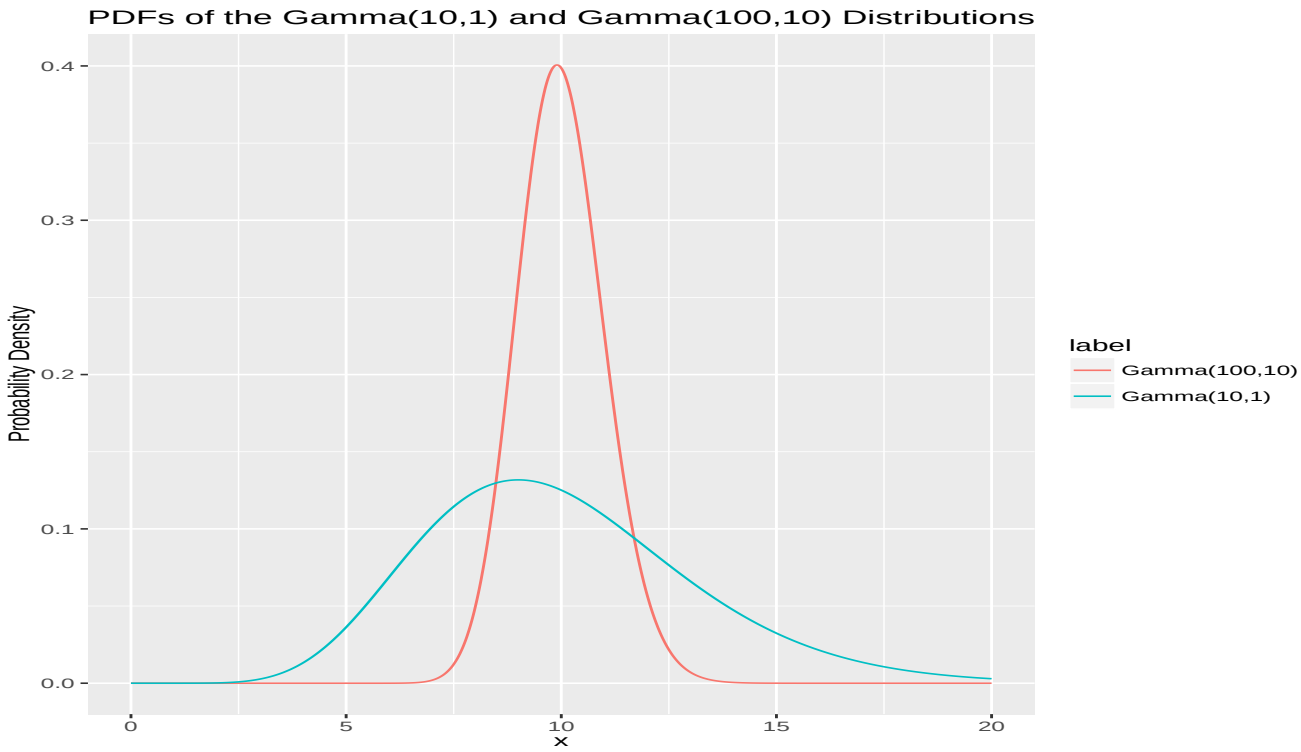
So now we look at the Gamma distribution.

```
x_seq <- seq(0, 20, by = 0.01)

gamma_010_01 <- dgamma(x_seq, shape =  10, rate =  1)
gamma_100_10 <- dgamma(x_seq, shape = 100, rate = 10)

plot_dt <- rbind(data.table(label   = 'Gamma(10,1)'
                           ,x       = x_seq
                           ,density = gamma_010_01)
                ,data.table(label   = 'Gamma(100,10)'
                           ,x       = x_seq
                           ,density = gamma_100_10))

ggplot(plot_dt) +
    geom_line(aes(x = x, y = density, colour = label)) +
    xlab(expression(x)) +
    ylab("Probability Density") +
    ggtitle("PDFs of the Gamma(10,1) and Gamma(100,10) Distributions")
```



Like the Beta distribution, the larger the size of the parameters, the tighter the distribution around the mean.

Exercise 6.1 The Stan file for the fully Bayesian model is in `binarytrial_full.stan`. Using the model specified in this file with the data contained in `binarytrial_twotest.rds` to sample from the prior and posterior distributions.

Exercise 6.2 How does using a fully Bayesian model change the inferences we make on the parameters of the model based on the data?

Exercise 6.3 Create a new dataset for five trials with different counts of trials for each test. Using different values of θ for each test, but have them be relatively similar.

Exercise 6.4 Run the fully Bayesian model on your data. Make inferences on the posterior distribution. How do these inferences match up with the knowledge you already have from generating the data?

Exercise 6.5 Create another dataset for the five tests. The dataset should be similar to the previous one, but this time use very different values for θ.

Exercise 6.6 Run the model with the new data. How do your inferences change between the two datasets, and does this make sense in light of how the data was generated?

Exercise 6.7 Use the supplied function The accompanying code file contain the function `generate_hierarchical_binomial_data()`. This function will generate binomial data for tests generated from a category with μ and κ, and for a varying number of tests and trials. Use this function to generate binomial

data for 5 tests and 50 trials and then for 50 tests and 5 trials. The code has been vectorised, so make sure you understand how it works.

Exercise 6.8 Use the full Bayesian model with both datasets to generate the posterior distributions for the various parameters associated with both datasets. Examine the posterior distributions to decide which dataset is better for making inferences on the mean value of the mint μ. Does your answer make sense? What conclusions can we draw about the tradeoff between the number of tests versus the trials per test in this case? How might this be generalised?

7. Switching Focus: Online Clickthrough Rates

Coins and coin tosses are a good place to start with binomial trials, but suffer from a major weakness. There is no such thing as a biased coin really. It is very hard to do. In many cases, manipulating the output of a coin toss is better served by practice at the mechanics of doing a coin flip.

For that reason, we will switch the topic of analysis: we will now analyse clickthrough rates of online ads. The click-through rate (CTR) for an ad is the ratio of the people that click on an ad versus the number of people that were shown the ad. CTRs vary wildly depending on the circumstance, and ranging from $1e-6$ (1 click per million views) to 0.2.

Switching focus allows us to do something more realistic and interesting, helping making everything more concrete in our minds. Our 'coins' now become a particular ad design, with multiple views and clicks of that design, and those designs are grouped into different products. The goal of the analysis is to get a sense of the effectiveness of the various ad designs and products, judging performance on the CTR.

While the methods employed in the previous section work fine, a more 'Bayesian' approach would be to incorporate the product data into the model, rather than running a separate run for each one. As you imagine, this is especially true if there are a large number of separate entities at the highest hierarchy of the model. Splitting the data by product will lead to a large number of models

Incorporating the product data for each ad is relatively simple — we just need to index by μ and κ for each product, as shown in Figure 2.

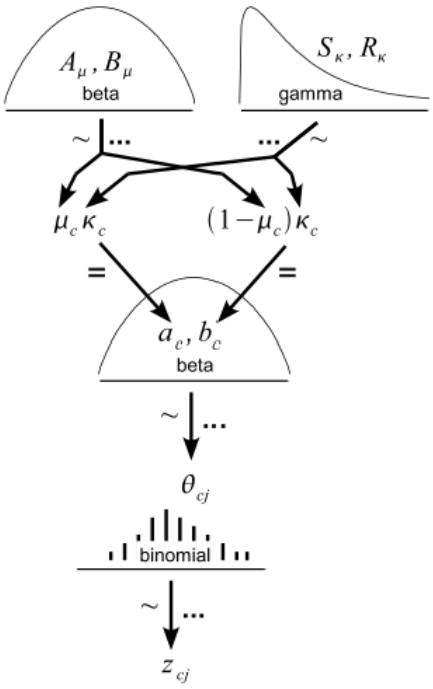


Figure 2: Graphical Representation of the Binomial Trials Hierarchical model

For the purposes of estimating and inferring on ad-quality, we are looking for ads that are as successful as

possible i.e. have a θ as close to 1.00 as possible, and can do so as consistently as possible, i.e. the variance of the θ values for the ads is as low as possible.

Exercise 7.1 Generate data using the supplied funtion `generate_multiple_hier_trial_data` for 5 products. You can choose your own input parameters or work with the supplied defaults.

Exercise 7.2 Use simple density estimation and summary statistics to get a quick estimate of the quality of the products.

Exercise 7.3 Use the supplied hierarchical Stan model to get estimates for the various μ and K for each product.

Exercise 7.4 Rerun the sampler, but now use the file in `multipleprod_lognormal_k.stan`, which puts a lognormal prior on the value of K. Compare the output to the previous model that uses a Gamma prior for K.

Exercise 7.5 Make a larger dataset, using more designs per product and more views per design, and rerun the samplers with this data. How does it affect the inferences?

Exercise 7.6 Vary both the number of designs per product and the number of views per design to determine which has the biggest effect on inference on K.

8. Posterior Predictive Checks

Posterior predictive checks (PPCs) are a way to use the output of the sampler to generate 'fake' data from your sample. You then compare the fake data to the original data and assess what aspects of the data are not being captured by the model.

It is probably no surprise that the use of PPCs is quite an art, and involves thinking carefully about what you are doing. Ask very high level questions about your problem and then see if there is a way to quantify that question. You then compare quantities from your data to the generated quantities produced by the Stan model.

Exercise 8.1 Using our hierarchical model, what quantities might help us assess the quality of the product ads? Think about ways in which we might encapsulate that in the data?

Exercise 8.2 Using the `generated quantities` block in Stan, use our latest model to generate samples of this value from our data. Compare the distribution of this generated quantity to that from our data.

Exercise 8.3 What can you infer from making this comparison?

Exercise 8.4 What other quantities can we think about to help us assess our current model?

9. Further Work and Open Questions

Exercise 9.1 Is there ways in which we can improve the data generation to help us understand how this approach handles differences?

Exercise 9.2 How do we incorporate errors in data collection to our model?

Exercise 9.3 What kind of effects would these errors have on our output and our conclusions?

Exercise 9.4 How can we extend this model at the higher levels?

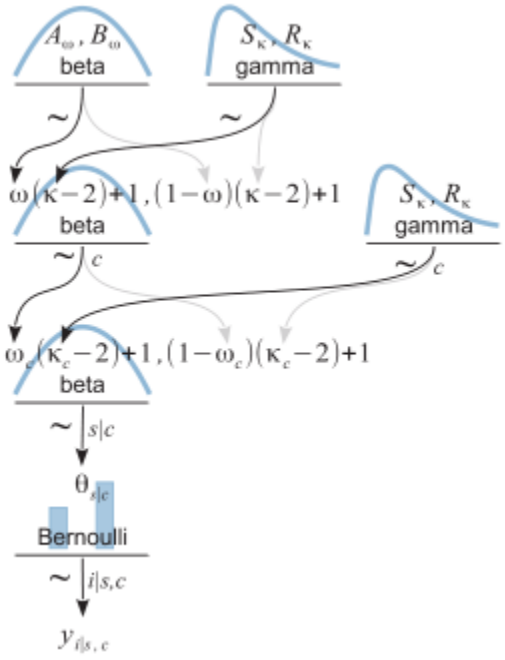


Figure 3: Graphical Representation of Hierarchical Model for Adding Hierarchies