Bayesian Statistics

Group exercise 1

Read before you start

For these group exercises, you are expected to deliver the following:

- R & JAGS code for each sub-exercise.
- A report (PDF) containing the answers to the questions, including plots/figures where necessary, and referring to the code you wrote. Make sure figures etc. are properly visible and have informative labels.

Apart from the book by Kruschke, you may also need the JAGS manual[1] to look up the details of the distributions you will need to use.

1 So... does it actually work?

Bayesian inference is about *model inversion*. You describe, in your model, how parameters generate data (i.e. $p(\text{data}|\text{parameters})$). By applying Bayes' rule you *invert* the model, and compute instead what the most likely parameters are, given the data (i.e. $p(\text{parameters}|\text{data})$). On real data, it can be quite challenging to make sure you did everything correct and got the right posterior distribution. However, when we generate data ourselves, we can use it to verify that the parameters we learn from this data, are actually those parameters that we used to generate the data with in the first place.

1.1 Recovering the correct model

In this exercise, you will explore basic models that generate data, and use JAGS to recover the parameters that you used to generate the data with. Use the R template `doesitwork_template.R` to write all your analyses in (it contains different blocks for every major change).

1. (20pts) The first model concerns data distributed according to a Gaussian distribution, i.e. $x_i|\mu, \sigma^2 \sim \text{Gaussian}(x_i|\mu, \sigma^2)$. We want to learn μ from $x_i, i = 1..N$. These x_i could for example reflect the position of an object on in an environment, or the numerical outcomes of an experiment. That means we need a prior for μ. Interestingly, for the prior on the mean of a Gaussian distribution we can use again a Gaussian distribution, so we have $\mu|\hat{\mu}, \hat{\sigma}^2 \sim \text{Gaussian}(\mu|\hat{\mu}, \hat{\sigma}^2)$. Together, they form this simple generative model:

$$\begin{aligned} \mu|\hat{\mu}, \hat{\sigma}^2 &\sim \text{Gaussian}(\mu|\hat{\mu}, \hat{\sigma}^2) \\ x_i|\mu, \sigma^2 &\sim \text{Gaussian}(x_i|\mu, \sigma^2) \qquad\qquad i = 1..N \ . \end{aligned}$$

   You are going to try to recover μ from data that you must now generate. Pick a value for μ, which will be the 'true' value. Draw $N = 10$ data points from a Gaussian distribution, i.e. create $\mathbf{x} = (x_1, \ldots, x_N)$. You may fix $\sigma = 2.0$.

2. (20pts) Now construct the JAGS model to *learn* the posterior distribution of μ, given the data you have just generated[2]. In this model, you now also need to specify your prior (the first line of the generative model). The parameters $\hat{\mu}$ and $\hat{\sigma}^2$ you have to decide for yourself. Try setting it far away from the true value of μ, so you can see whether the data convinces the

[1] http://www.stat.yale.edu/ jtc5/238/materials/jags-2.1.0-user-manual.pdf

[2] JAGS uses a slightly different convention to parameterize the Gaussian distribution. Instead of supplying the variance, you must supply the *precision*, which is $1/\sigma^2$.

model to 'move away from the prior'. For this simple model, `n.iter=10000` easily is sufficient samples. Note that you can either specify the hyperparameters in R, and pass them to the `jags.model` function as if they were data, or you can define them in the JAGS code itself.

3. (10pts) Run the JAGS model and plot the posterior distribution (use the `plotPost` function). Write down the lower and higher bound of the Highest Density Interval. Repeat this procedure for $N = 100$ and $N = 1000$. What does the change in the HDI indicate?

4. (20pts) A slightly more elaborate model also learns the width of the Gaussian distribution, i.e. its variance σ^2 (or the standard deviation σ). Standard deviation is a continuous number and is always equal or greater than zero. This can be modeled using a Gamma distribution, so we can extend the generative model as follows:

$$\begin{aligned}\sigma|a,b &\sim \text{Gamma}(\sigma|a,b)\\ \mu|\hat{\mu},\hat{\sigma}^2 &\sim \text{Gaussian}(\mu|\hat{\mu},\hat{\sigma}^2)\\ x_i|\mu,\sigma^2 &\sim \text{Gaussian}(x_i|\mu,\sigma^2) \qquad\qquad i = 1..N\ .\end{aligned}$$

   In the next code block, create a new JAGS script that can infer both μ and σ. Is your model still able to reasonably well recover the parameters? Write down the HDI bounds for both μ and σ for $N = 10$, $N = 100$ and $N = 1000$. Make sure you are still using the same values for μ and σ, or it will be impossible to see whether the recovery has worked.

5. (10pts) Using the posterior means for μ and σ, you can plot the Gaussian probability density function using the `plotGaussian(data, mean, sd)` function (provided in the script template). Do you think the distribution matches the data? Do you see a difference if you increase/decrease the number of generated data points?

1.2 Recovery with the wrong model

So far you have worked in the ideal case where the model that you use for Bayesian inference is actually the model that has generated the data. But typically, we don't know this model. We now explore the (unfortunately realistic) scenario where you have assumed a particular model, while knowing fully well that it probably doesn't capture all properties of your data.

1. (10pts) Change your data generation procedure to generate data from a *student-t* distribution instead of a Gaussian. This distribution is known for its *fat tails*, which means that values far away from the mean have a higher probability than in the Gaussian distribution. In R, you only have to supply the `df` parameter to the function that generates student-t random numbers (as well as `n`, the number of samples you want). Set `df=2`. Generate $N = 1000$ data points and plot the histogram. Compared to the earlier subquestions, your data should now look like it contains some outliers – values far from the mean.

2. (10pts) Now run the same model you used earlier to estimate μ and σ^2, on this data and plot the fitted Gaussian distribution over the *student-t* distributed data. Compared to the application of the model on actual Gaussian data, what difference do you see?

2 Modeling and parameter learning for exam results

In this exercise, we'll use some very small data sets and Bayesian modeling to actually answer some basic research questions. Once again, we'll study the performance of students on multiple choice exams (us teachers are obsessed with these type of examples). This time however, you'll be using JAGS to construct increasingly complex models for such data.

2.1 To study or not to study

Suppose we have an exam of $n = 40$ true-or-false questions, and $p = 15$ students that take this test. We observe the following p scores on the exam:

$$\mathbf{k} = (19, 20, 16, 23, 22, 30, 38, 29, 34, 35, 35, 32, 37, 36, 33)$$

correct answers. If you look closely at this data, it seems as if it captures two different 'regimes' of students: one where the students have approximately chance-level performance and get roughly $n/2$ of the questions correct, and one where they have at least some understanding of the course material. We would like to know to which group each student i belongs, and what the success rate (= probability of a correct answer) is for either group.

The first step in any Bayesian modeling application is to construct the graphical model that formalizes this problem and allows you to think about how the relevant variables interact. This model should include:

- k_i (observed): the number of correct answers for student i.
- n (observed): the total number of questions.
- z_i (latent): the group label for student i (either 0 (guessing students) or 1 (studying students). For now, assume the probability of a student belonging to either group is 50/50, i.e. $z_i \sim \text{Bernoulli}(0.5)$.
- ϕ (latent): the probability of a correct answer for a student who studied.
- ψ (hyperparameter): the probability of a correct answer for a student who guessed.

You may also have to come up with a few variables yourself.

Note: each exam question is essentially a coin flip. The distribution that models *the number of successes* is known as the *binomial* distribution, and will be useful here. Its parameters are the total number of questions as well as the probability of having a correct answer. Look at its Wikipedia page for more information.

1. (20pts) Draw the *graphical* and write down the corresponding *generative* model (cf. Lecture 4). Think about what distribution makes sense for each variable, how these variables interact, and which variables are missing for the model to work.
2. (10pts) Using your generative model, describe how you can generate a random vector like $\mathbf{k}$ (but of course with different numbers since it is random) using your model.
3. (20pts) Implement the model in R & JAGS.[3] Use the template file `examresults_template.R` as a basis for your script.
4. (10pts) Run the model. Make sure the sampler has *converged* (use the plotting functions discussed in Lecture 5 and the book chapter on JAGS) before proceeding. Indicate for how many iterations you have let your sampler run and why you think this is enough.
5. (10pts) Using the approximated posterior distribution, what is the (mean of) the probability of a correct answer, for students who studied? What is the standard deviation around this mean?
6. (10pts) What labels does your model assign to the students? Which students studied, and which did not?
7. (10pts) How can you extend the model to learn the probability of a student having studied? Show the extended graphical and generative models.

2.2 Individual variation using an hierarchical model

Using the model you constructed for the first subquestion of the previous question, you inferred the success rate of the group of students who studied. But the model makes a strong, and most likely wrong, assumption: it assumes that the people in the group that studied, all have the *same* probability of answering a question correctly.

1. (10pts) Which parameter(s) in your model captured this assumption?

[3]You can implement an `if...then...else...` statement in JAGS using for example `equals(z[i],0)*a + equals(z[i],1)*b`, which gives `a` if `z[i]==0` and `b` if `z[i]==1`.

2. (10pts) Modify your model (both the graphical model as well as the generative model) to capture that students who studied still have individual variability in their success rates (but all of these have a success rate larger than 0.5, otherwise they would belong to the guessing group).[4]

3. (20pts) Implement the model using JAGS. Use the template file `examresults_template.R` as a basis for your script. What are the probabilities of having a correct answer, for all students who studied?

2.3 Easy and tough exam questions

Of course, on a typical exam, not only will some students have studied better than others, but some questions will also be harder than others. We can express this using $\theta_{ij} = p_i q_j$, where p_i is the probability person i has a question correct and q_j is the probability question j gets answered correctly, and finally θ_{ij} being the probability that person i answers question j correctly.

1. (20pts) Look at the R script template for this exercise in the `examresults_template.R` file. Note that the data is provided as a matrix $\mathbf{k}$. Write down the graphical and generative model for this situation. Assume flat priors on p_i and q_j.

2. (20pts) Implement the model in JAGS, again using the R template. Using this implementation, who are the three students that studied best? And which three studied worst?

3. (20pts) In Bayesian inference, some variables are observed (here the values k_{ij}, some are latent and inferred, and some are hyperparameters. But the principle doesn't care which variable is which. We can use this to predict the result for answers that were not even observed (for example, some answers were illegible due to bad handwriting). In R, a missing value can be denoted as `NA` (not available). Set the answers $k_{1,13}$, $k_{8,5}$ and $k_{10,18}$ to `NA`, and rerun the model. What are the predicted (expected) outcomes for these missing observations? How do these predictions change if you change your flat prior on the question correctness probabilities q_j to a prior that expresses that the questions are very difficult?

2.4 Inferring a difference

Besides wanting to learn parameters, we often want to use (Bayesian) statistics as a basis to make decisions from. Relevant research questions often ask whether there is a difference between some parameter for one group and the same parameter in another group. For example, we might want to find out whether one batch of students did better on a course than another batch. Note that these batches don't have to be of equal size.

Where the *Bernoulli* distribution captures the outcomes of individual students (pass or fail), the *binomial* distribution captures the number of students, out of the total number in the batch, who passed or failed. The binomial distribution has a parameter θ (conceptually the same as the Bernoulli parameter) and n, which is the total number of trials. You can look up the Wikipedia page on the binomial distribution for more details and some examples

Assume we observe $k_1 = 37$, $n_1 = 50$ and $k_2 = 48$, $n_2 = 63$, indicating for two groups of students how many passed the course. Both groups have a latent variable indicating the probability of a student passing (e.g. θ_1 and θ_2). Next to that, there is a latent variable δ that captures the *difference* between θ_1 and θ_2. Note that δ is a *deterministic* variable, not a stochastic one!

1. (20pts) Write down the graphical and generative models for this context. Think about the direction of the arrows and how you express the deterministic variable δ.

2. (20pts) Now implement the model in JAGS, using again the `examresults_template.R` file, and plot the posterior distribution of δ. Would you, based on this result, conclude that there is a difference between the groups? Does your answer change if $n_2 = 49$?

[4]Hint: obviously, the prior on a success rate is the beta distribution. However, sometimes it is easier to specify the beta distribution in terms of its *mean* μ and the certainty around the mean, κ. In JAGS, you then write `a <- mu*kappa`, `b <- (1-mu)*kappa` and `x ~ dbeta(a, b)`. This way, you can think about a mean of a distribution rather than pseudo-counts.