Class 2: Likelihood and Bayesian imputation methods

Andrew Parnell
andrew.parnell@mu.ie

PRESS RECORD https://andrewcparnell.github.io/mda_course

In this class . . .

- Revision of likelihood and Bayes
- Imputation via Bayesian inference
- Approximate methods for imputation
- The `mice` package/algorithm

Revision of likelihood

- If we are in a complete data situation we need to compute $P(Y|\theta)$ where Y are our observations, and θ is the parameter(s) of our model. This is the **likelihood**
- If we assume that the data are (conditionally) independent we have $P(Y|\theta) = \prod_{i=1}^{n} P(Y_i|\theta)$
- In linear regression we split the data up into a response Y and covariates X, and θ would now be our intercept, our slope, and our residual standard deviation
- We now find $\prod_{i=1}^{n} P(Y_i|X_i, \theta)$
- To get this to work we need to choose a probability distribution for P. For linear regression this is a normal distribution

Calculating the likelihood

For the Whiteside data

```
library(MASS)
y = whiteside$Gas
x = whiteside$Temp
prod(dnorm(y, 5.5 - 0.3*x, 1))
```

```
## [1] 8.780779e-32
```

This is pretty small so most people work with the log likelihood to keep numerical stability

```
sum(dnorm(y, mean = 5.5 - 0.3*x, sd = 1, log = TRUE))
```

```
## [1] -71.51016
```

Maximising the likelihood

- One way to find the 'best' parameters is to try lots and lots of different values and take the ones that provide the biggest log likelihood. This is very inefficient
- Another way is to use mathematics - we can often maximise the likelihood using calculus. (We are not going to do this)
- R has a number of very efficient built-in optimisation routines (e.g. `optim`) which will find the best values for us

Likelihood inference for missing data

- Now suppose we have a missing observation:

```
y[5] = NA
```

- Our parameters are now $\theta = (\alpha, \beta, \sigma, x_5)$. In R we could optimise via:

```
nll = function(theta) {
  newy = y; newy[5] = theta[1]
  -sum(dnorm(newy, mean = theta[2] + theta[3]*x, sd = theta[4], log = TRUE))
}
answer = nlminb(rep(2, 4), nll, lower = c(-Inf, -Inf, -Inf, 0))
print(answer$par)
```

```
## [1]  4.6122369  5.4374802 -0.2845665  0.8304745
```

Did it work?

```
plot(x, y)
points(x[5], answer$par[1], col = 'red')
```

Bayes' theorem

- Treating the missing data as parameters and maximising the likelihood works fine for large data sets with low degrees of missingness
- To get uncertainties we rely on large sample approximations which rely on asymptotic normality. This sometimes leads to inappropriate confidence intervals
- Frequentist inference also often relies on p-values which are easily gamed.
- There is a saviour to the rescue...

Enter Bayes

An essay towards solving a problem on the doctrine of chances (1763)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

How does Bayesian statistics work?

- Bayesian statistics is based on an interpretation of Bayes' theorem
- All quantities are divided up into *data* (i.e. things which have been observed) and *parameters* (i.e. things which haven't been observed)
- We use Bayes' interpretation of the theorem to get the *posterior probability distribution*, the probability of the unobserved given the observed
- Bayes' equation is usually written mathematically as:

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

This reads as posterior is proportional to prior times likelihood

Choosing a prior

There are several choices when it comes to specifying prior distributions:

- *Informative*, when there is information from a previous study, or other good external source, e.g $\theta \sim N(-30, 1.5^2)$
- *Vague*, when there is only weak information, perhaps as to the likely range of the parameter e.g. $\theta \sim N(0, 10^2)$
- *Flat*, when there is no information at all about a parameter (very rare).

In most cases, priors can only be understood in the context of the data, for example a $N(0, 100)$ prior might be uninformative

Fitting a Bayesian model

- The first step is to choose the prior and likelihood probability distributions
- Then we fit the model to obtain a posterior distribution
- These posterior distributions appear as *samples* from the posterior probability distribution rather than direct estimates of means/standard deviations or equations
- We can thus obtain complicated probability distributions for our parameters which do not need to be asymptotically normal

Gibbs sampling

- Fitting a complicated model to get a posterior distribution is hard, and is not usually possible using maximisation techniques
- A useful trick is to use a Gibbs sampler where we sample each parameter in turn conditional on the others
- If we have set of parameters $\theta = (\theta_1, \ldots, \theta_k)$, then we create:

$$P(\theta_j|\theta^{(-j)}) \propto P(Y|\theta)P(\theta_j)$$

- Quite often the probability distributions simplify depending on the priors and the likelihoods

- Even when they simplification isn't possible there are other techniques to simulate from these probability distributions
- The algorithm works with starting guesses for θ, then iterates through all the parameters over and over again and, theoretically, is guaranteed to end up at the posterior distribution.

Fitting a Bayesian linear regression model in JAGS

```
model_code ='
model {
  # Likelihood
  for(i in 1:N) {
    y[i] ~ dnorm(intercept + slope*x[i], residual_sd^-2)
  }
  # Priors
  intercept ~ dnorm(0,10^-2)
  slope ~ dnorm(0,10^-2)
  residual_sd ~ dunif(0,10)
}
'
```

Running the model

```
library(R2jags)
```

```
## Loading required package: rjags
## Loading required package: coda
## Linked to JAGS 4.3.0
## Loaded modules: basemod,bugs
##
## Attaching package: 'R2jags'
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
model_parameters =  c("intercept", "slope", "residual_sd")
model_run = jags(data = list(N = length(y), y = y, x = x),
                 parameters.to.save = model_parameters,
                 model.file = textConnection(model_code))
```

```
## module glm loaded
## Compiling model graph
```

```
print(model_run)
```

```
## Inference for Bugs model at "4", fi
##  3 chains, each with 2000 iteration
##  n.sims = 3000 iterations saved
##              mu.vect sd.vect    2.5%
## intercept      5.424   0.241   4.943
## residual_sd    0.873   0.088   0.724
## slope         -0.282   0.042  -0.363
## deviance     139.722   2.454 136.867
##
## For each parameter, n.eff is a cruc
## and Rhat is the potential scale rec
##
## DIC info (using the rule, pD = var(
## pD = 3.0 and DIC = 142.7
## DIC is an estimate of expected prec
```

Bayesian computation with JAGS and Stan

- JAGS (Just Another Gibbs Sampler) fits Bayesian models using Gibbs Sampling to update the parameters in turn
- It has its own language and so the code needs to be stored in a text string or a separate file
- It's usually pretty fast and can run models with many parameters
- A related language, Stan, is slightly more fashionable but a bit slower to run (at first) and harder to teach! (But also worth looking at for missing data analysis)
- We will do some missing data analysis with JAGS in the computation session

An example of JAGS code for missing data

```
model_code = '
model
{
  # Likelihood
  for (i in 1:n) {
    y[i] ~ dnorm(intercept + slope * x[i], residual_sd^-2)
  }
  # Priors for the missing x values
  for(k in 1:n_miss_x) {
    x[miss[k]] ~ dunif(min_x, max_x)
  }

  # Priors
  intercept ~ dnorm(0, 100^-2)
  slope ~ dnorm(0, 100^-2)
  residual_sd ~ dunif(0, 100)
}
```

Why doesn't everybody use Bayesian inference for MDA?

- This approach works well for small amounts of missing data, and/or when there is good prior information on what those missing values are
- It doesn't work so well for larger missing data sets as the parameter gets too big and the models get very slow or even break
- Instead what people tend to do is use an approximation; use a Bayesian model to impute the missing values, then fit the model using these imputed values as though they were the prior
- This is a bit of a cheat, but it does seem to work well in simulation studies

Imputing first, model later

- If we take the more practical route of imputing first and modelling later, we can still use the tools of Bayesian inference
- We now treat our observed and missing data as though it was a multivariate probability distribution and try to learn the parameters
- If all the data are continuous, a common distribution to choose is the multivariate normal distribution

$$Y_i \sim N(\mu, \Sigma)$$

  where Y_i is the vector of observations, μ a vector of means, and Σ the covariance matrix
- With missing data this can be a really hard likelihood to right down

Multivariate imputation with ignorable monotone missingness

- Recall that missingness mechanism is ignorable if we don't need to include a model for the M missingness matrix
- Also recall that monotone missing means that the variables $Y_{i1}, \ldots, Y_{ip}$ can be ordered such that missingness occurs in a staircase pattern; once a variable is missing on variable j it is also missing for variables $j + 1, \ldots, p$
- Let's look at an example

Ignorable monotone missingness example

- Here's a 2D example (which will always satisfy monotone missingness):

```
whiteside3 = whiteside[,2:3]
whiteside3[35:56,2] = NA # Last 22 values on Gas are missing
```

- The job now is just to predict those missing values. We need a likelihood $P(Y_{\text{obs}}|Y_{\text{mis}}, \mu, \Sigma)$ but this is hard to write down, e.g.

```
library(mvtnorm)
sum(dmvnorm(whiteside3, mean  = rep(5,2), sigma = diag(2), log = TRUE))
[1] NA
```

- But we can use our trick from earlier! Let Y_1 and Y_2 be the two columns:

$$P(Y_1, Y_2|\mu, \Sigma) = P(Y_2|Y_1, \mu, \Sigma) \times P(Y_1|\mu, \Sigma)$$

The first term we can write using our conditioning formula (see last class), and the second has no missing values!

JAGS code for the ignorable monotone missingness case

```
model_code = '
model
{
  # Likelihood
  for (i in 1:N) {
    y1[i] ~ dnorm(mu1, sd1^-2)
  }
  for (i in 1:N2) {
    y2[i] ~ dnorm(mu2 + (sd2/sd1)*rho*(y1[i] - mu1), ((1-rho^2)*(sd1^2))^-1
  }

  # Priors
  mu1 ~ dnorm(0, 100^-2)
  mu2 ~ dnorm(0, 100^-2)
  sd1 ~ dunif(0, 100)
  sd2 ~ dunif(0, 100)
  rho ~ dunif(-1, 1)
```

An alternative to Gibbs: the EM algorithm

Overview

Why the EM algorithm works

Example

Why is Bayes impractical for imputation?

An alternative: the mice approach

The mice algorithm

Sensitivity checking

How mice works

Pooling models using mice