Bayesian Networks I

David Hinrichs

11/12/2017

Intro

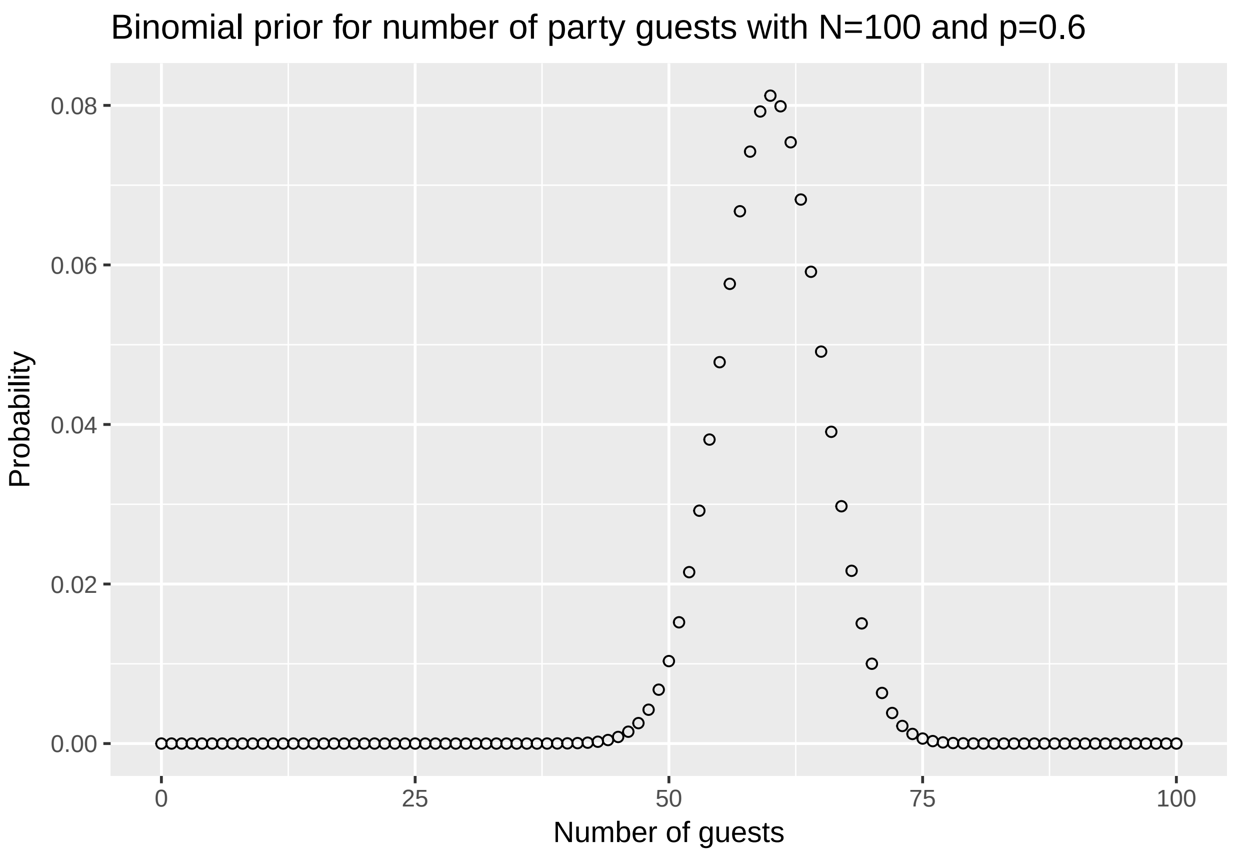
This gives an example of how to use JAGS to perform sampling in context of bayesian networks. But first, some preliminaries. We will load 'ggplot' and the 'rjags' package that lets us talk with JAGS. Installing rjags is pretty straightforward on MacOS and Windows, but seems to involve some work on Windows.

```r
require(ggplot2)
require(rjags)
```

Setting

Remmember our situation: We came home to find our apartment in disarray with 132 beer bottles scattered throughout. Our housemates obviously had a party and we want to know how many guests they had. We are sure the Facebook event only contained N=100 people and that on average people accept such invitations with a prob. between .5 and .7. Further we assume that the number of beers each person drank is geometrically distributed with stopping parameter lambda=0.5.

Our goal is to infer the number of guests at a party from reasonable assumptions and observed data. More specifically, we want to use Bayes theorem and JAGS to update our prior for the number of guests at the part via the observed data of the number of beer bottles they drank. Starting with our prior, we look at the binomial distribution in the case Binom(100,0.6):



This is our current best guess for how many people attended the party. We will use our data and further asumptions to get a better answer.

Working with JAGS

Now we convert our model from the black board into a text file and pass it onto JAGS together with our observations. The formulation is simple and straightforward - each arrow in the bayesian network that we drew on the board now becomes a line in the model file. Note that assignments of fixed parameter values are done using '<-' and probabilistic relationships are expressed via '~'

```
# define the model
modelString = "
model {
  B ~ dnegbin(lambda, G)
  p ~ dunif(0.5, 0.7)
  G ~ dbin(p,N)
  N <- 100
  lambda <- 0.5
}
"

# write the model into a text file called beer1
writeLines( modelString , con="beer1.txt" )
```

After the model has been defined we need to tell JAGS to run it against the data we have observed. The data can be encoded in a basic list and be passed inside the jags.model command. We specify three chains so we can later examine their differences for problems in the convergence of the model toward the limiting

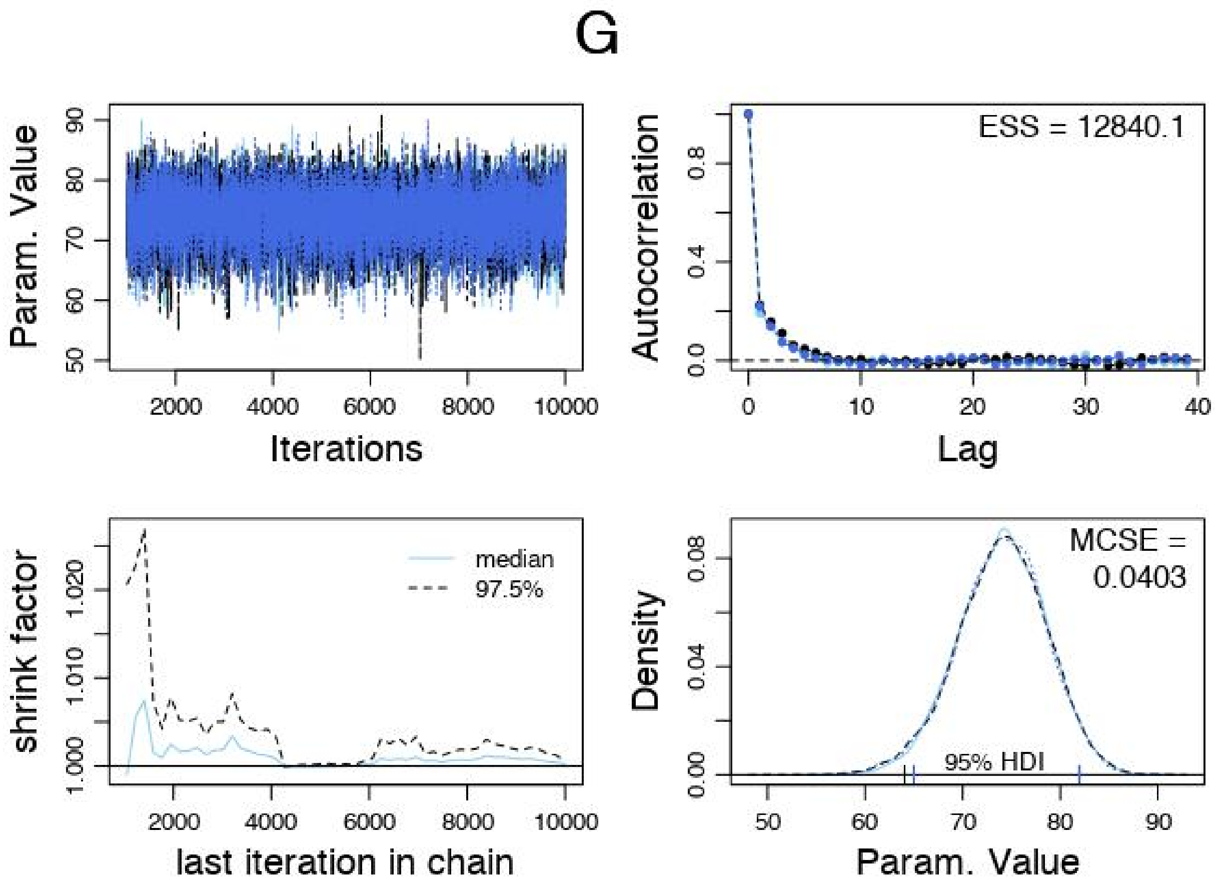


Figure 1: Diagnostics for the markov chains

distribution. Then we let JAGS run the model for 1000 steps to let the sampling distribution approach the limiting distribution before we start recording values. Finally we sample 9000 values from each of the three chains.

```
# Observation data of 132 beer bottles strewn across the apartment floor
dataList <- list(B=132)

# setup the model from our model file, give it data, specify the number of chains and let it run for 10
jagsModel = jags.model( file="beer1.txt" , data=dataList, n.chains=3 , n.adapt=1000 )

# pull 9000 samples across 3 chains from the posterior of the model, observe the values for the number
codaSamples = coda.samples( jagsModel, variable.names=c("G", "B"), n.iter=9000 )
```

Now that we have succesfully run the model we need to examine the output. This happens in two stages: At first we will take a look at the markov chains and afterwards we deal with the sampled values for our posterior for the number of guests at the party.

```
# MCMC diagnostics for the markov chains
source('./DBDA2Eprograms/DBDA2E-utilities.R')
```

```
diagMCMC(codaSamples, 'G', saveName = 'beer', saveType = 'jpg')
```

Top left: Values per iteration

Top right: Autocorrelation against distance in the chains

Bottom left: Gelman-Rubin convergence measure via total variance divided by in-chain variance

Bottom right: density plot for observed parameter

Since we are satisfied with the diagnostics of the markov chains, we will regard their sampled values as actually useful and move on to examine them.

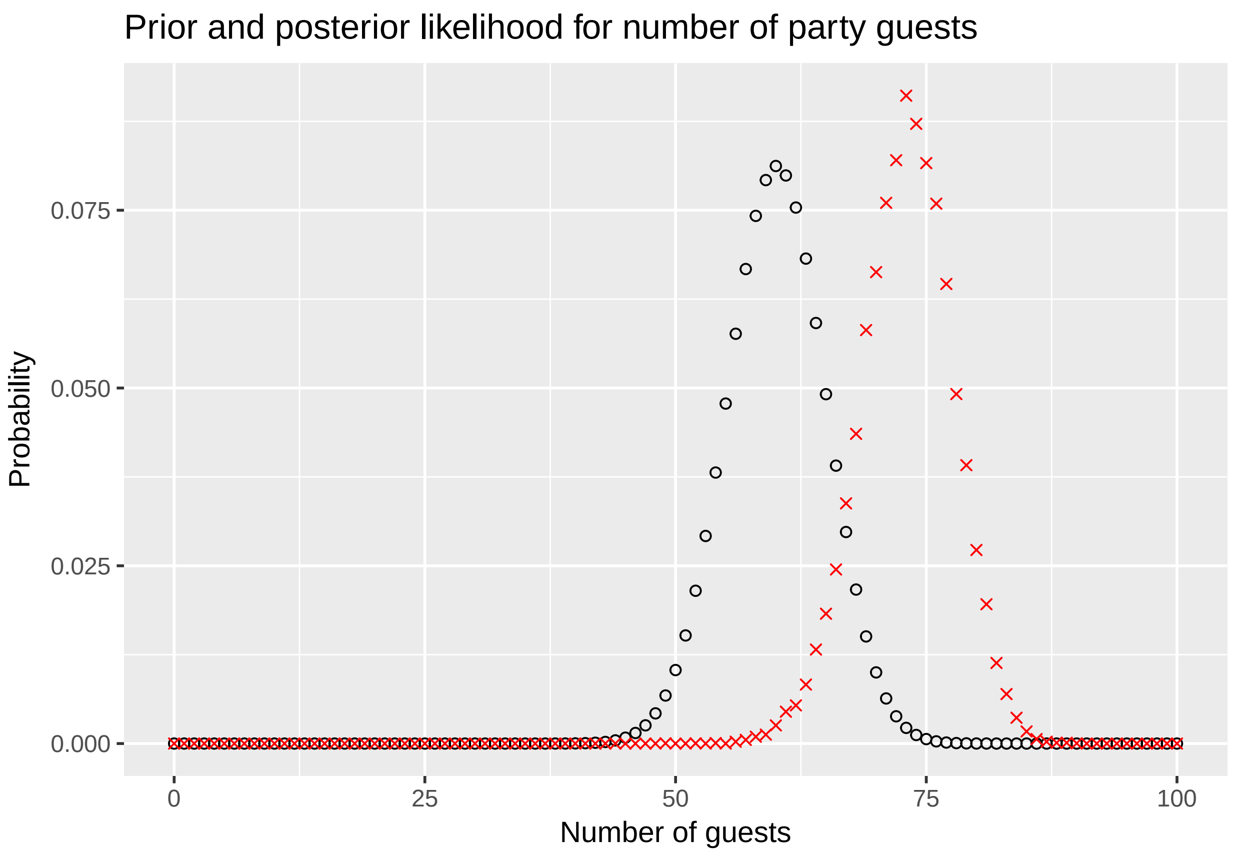
Results

Now we can take a look at the summary of observed values for the number of beer bottles (just as an example, since this is fixed) and the number of guests at the party.

```
summary(codaSamples)
```

```
##
## Iterations = 1001:10000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 9000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##     Mean    SD Naive SE Time-series SE
## B 132.00 0.000  0.00000        0.00000
## G  74.01 4.564  0.02777        0.04033
##
## 2. Quantiles for each variable:
##
##   2.5% 25% 50% 75% 97.5%
## B  132 132 132 132   132
## G   65  71  74  77    82
```

And finally we use our sampled data to build a posterior distribution for the number of guests at the party given our observation of the beer bottles left on the next morning. We have foregone to plot a histogram since in the case of discrete distributions the histogram is just a less pretty density plot.



So the beer bottles we found suggest that our housemates had a bigger party than we thought. This calls for a WG-meeting!

Conclusion

Initially we started with the prior distribution (black). Then we formulated a graphical model that contained observable data (# bottles). Using JAGS we where able to perform sampling from the posterior distribution without having to solve the integral that appeared in the denominator of Bayes' theorem. The reward for our work is a posterior distribution (red) that gives us reason to believe there where more people at the party than we initially expected.