Mathematics in Machine Learning: Theoretical Tesina

Andrea Cognolato, s281940@studenti.polito.it

July 2021

1 Introduction

1.1 Probabilistic Programming

Probabilistic programming is a programming paradigm where the users can freely specify probabilistic models and reason about them without having to manually implement their inference.

A probabilistic model is set of stochastic and deterministic relationships between variables, such as:

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2)$$
$$\mu_i = \beta_0 + x_i \beta_1$$

Given this model and some data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ a user could ask questions like:

- What are the most likely values for β_0, β_1?
- What's an interval in which β_1 lies, with 95% confidence?
- Are β_0 and β_1 correlated?

By embracing a Bayesian approach we can devise automatic ways to get an answer to these questions, without having to perform tedious manual calculations or having to manually write a sampler.

1.2 Computational Bayes

Under the Bayesian framework we to treat our parameters as random variables. The uncertainty on these parameters, prior to seeing any data, will be represented using a **prior distribution**. In particular, we can decide to model the uncertainty on the intercept β_0 and slope β_1 like this:

$$\beta_0 \sim \mathcal{N}(0, 10^5)$$
$$\beta_1 \sim \mathcal{N}(0, 10^5)$$

Let $\theta = [\beta_0, \beta_1]^T$ denote the complete set of parameters. Then can we define the prior distribution on our parameters:

$$\pi(\theta) = \pi(\beta_0, \beta_1) = f_{\beta_0, \beta_1}(b_0, b_1) = P(\beta_0 = b_0, \beta_1 = b_1)$$

We are interested in the **posterior distribution** $\pi(\theta \mid \mathcal{D})$, that is, how has the belief on our parameters changed after having observed some data.

Bayes theorem gives us an automatic way to obtain, at least up to proportionality, the posterior:

$$\pi(\theta \mid \mathcal{D}) \ \propto \ \pi(\theta)\mathcal{L}(\mathcal{D} \mid \theta)$$

Unfortunately, computing the posterior symbolically can be intractable. To sidestep this limitation, we can restrict ourselves to only drawing **samples** from the posterior, a tasks for which many scalable (w.r.t parameters and data) algorithms exist.

Sampling from a distribution is analogous to knowing it thanks to the Law of Large Numbers (LLN).

Let X be a random variable and let $\{x_i\}_{i=1}^n$ be i.i.d. (independent and identically distributed) samples. Then the LLN states that, in probability, the sample average converges to the expected value:

$$\frac{1}{n}\sum_{i=1}^{n} x_i \longrightarrow E(X)$$

1.3 Gibbs sampling

Now that we know that sampling from the posterior distribution is good enough to answer most of our questions, let us focus on how to actually extract samples from it.

For very simple distributions like uniform, normal, or exponential, many algorithms such as inverse transform sampling or rejection sampling are sufficient. However, when moving to complicated high-dimensional distributions, these methods do not scale well.

Gibbs sampling is a Markov chain Monte Carlo (MCMC) algorithm for obtaining samples from a distribution, when direct sampling is difficult. As with many MCMC methods the samples returned by the algorithm are not i.i.d., in fact, they are highly correlated. Thanks to techniques such as burn-in and thinning we can reduce this issue and work freely with our samples.

Gibbs sampling is particularly useful when we cannot sample directly from the joint distribution but we can easily sample from the **conditional distribution** of each variable.

1.3.1 Algorithm

Let k be the number of samples we want to get from a n-dimensional random variable $\boldsymbol{X} = (x_1, ..., x_n)$.

Let $\boldsymbol{X}$ be distributed with the joint PDF $p(x_1, ..., x_n)$.

We denote the ith sample by $\boldsymbol{x}^{(i)} = (x_1^{(i)}, ..., x_n^{(i)})$

The algorithm is the following:

- Begin with some initial value $\boldsymbol{x}^{(i)}$
- To get the next sample $\boldsymbol{x}^{(i+1)} = (x_1^{(i+1)}, ..., x_n^{(i+1)})$ we sample each component $x_j^{(i+1)}$ separately.
  - First we obtain x_1^{i+1} sampling from $X_1 \mid X_2 = x_2^{(i)}, ..., X_n = x_n^{(i)}$
  - We continue this cycling scheme by sampling x_2^{i+1} from $X_2 \mid X_1 = x_1^{(i+1)}, X_3 = x_3^{(i)}, ..., X_n = x_n^{(i)}$
  - Until we get $x_n^{(i+1)}$ from $X_n \mid X_1 = x_1^{(i+1)}, ..., X_{n-1} = x_{n-1}^{(i+1)}$
- This process is repeated k times

Because each dimension is explored separately, we can imagine Gibbs sampling as exploring the parameter space moving as if it were a chess board, as shown in figure 1.

1.4 JAGS

JAGS, acronym for Just Another Gibbs Sampler, is a software package for specifying probabilistic models and analyzing them using MCMC simulation. It is the spiritual successor to BUGS, WinBUGS[3], and OpenBUGS.

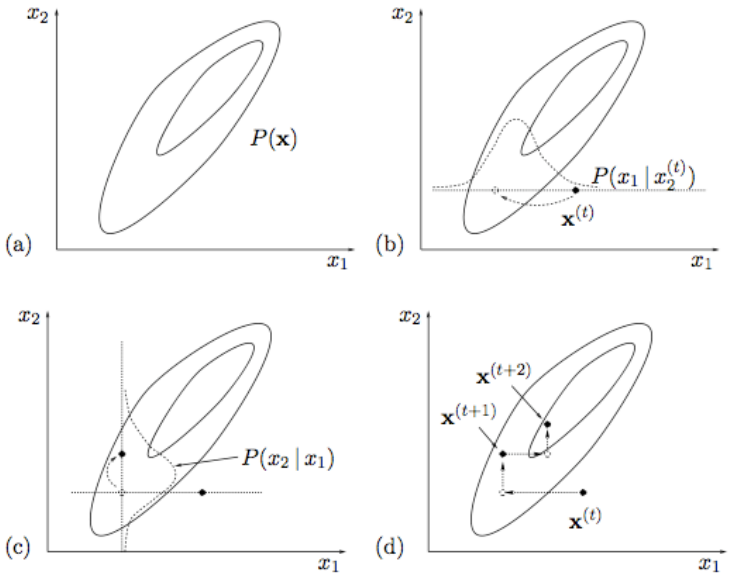

Figure 1: Visualizing how a Gibbs sampler explores a 2D parameter space. Since every dimension is explored separately, all steps will be axis-aligned, leading to its peculiar chessboard-like pattern. Taken from [4]

JAGS allows us to specify a probabilistic models in the BUGS language. For example, our linear regression model is specified like this:

```
model {
   beta0 ~ dnorm(0, 0.00001)
   beta1 ~ dnorm(0, 0.00001)
   tau <-  1 / sigma2
   for (i in 1:N) {
      mu[i] <- beta0 + x[i] * beta1
      y[k] ~ dnorm(mu[i], tau)
   }
}
```

Notice how instead of parameterizing the normal distribution using the variance σ^2 we use its inverse τ, known as the precision.

After writing the model in the BUGS language, JAGS will parse it and build a Directed Acyclic Graph (DAG) using the distributions and deterministic values as nodes, and their stochastic or deterministic relations as edges.

Thanks to the DAG, JAGS can infer the independence structure of our distribution and automatically obtain their conditional distributions. Additionally, using this knowledge, it can perform many optimizations such as conjugacy-detection or blocked-Gibbs-sampling. We can see an example of one of these DAGs in figure 2

2 Comparison of linear regression methods

Having described probabilistic programming and Gibbs sampling, let's now put it in practice. We are analyzing one of the Classic BUGS examples called Stacks. This example is taken from David Birkes and Yadolah Dodge's Alternative Methods of Regression, Chapter 9, Section 9.3 Example 3 [2].

The aim of this example is to explore how different error structures influence the results of linear regression.

The dataset is the famous stack loss dataset. It consists of 21 days of measurements from a factory for the

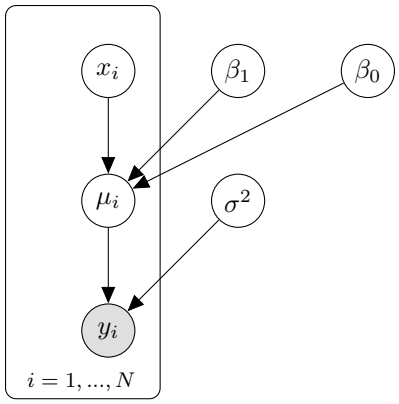

Figure 2: The DAG structure which our linear regression model encodes. The iteration on the index i is expressed using a *plate*, i.e. the rectangle around x_i, μ_i, y_i

oxidation of ammonia to nitric acid. Each measurement records the air flow x_1, the cooling water temperature x_2, the concentration of acid x_3, and the amount of ammonia that escaped before being oxidized, called stack loss y.

2.1 Theory

2.1.1 Linear regression as optimization

Linear regression, in its simplest form, is the problem of finding the intercept and slope of a line which best fits a set of points. That is, solving the following optimization problem:

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where

$$\hat{y}_i = \beta_0 + x_i\beta_1$$

or, written in matrix notation:

$$\min_{\boldsymbol{\beta}} ||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2$$

which leads to the solution:

$$\boldsymbol{\beta} = (X^T X)^{-1} X^T Y$$

2.1.2 Linear regression as probability

A probabilistic approach to linear regression is typically to model the errors as independent heteroschedastic and normally distributed (encoded by the $\sigma^2 I$ covariance matrix):

$$\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 I)$$

And it's possible to prove that maximizing the log-likelikood is equivalent to solving the aforementioned optimization problem, as the normal log PDF is basically the sum of squared errors:

$$\max_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{Y}) =$$

$$\max_{\boldsymbol{\beta}} \log \mathcal{L}(\boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{Y}) =$$

$$\max_{\boldsymbol{\beta}} -\frac{1}{2}||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 =$$

$$\min_{\boldsymbol{\beta}} ||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2$$

2.1.3 Different flavors of regression

But we may want to vary this model. Maybe our errors are not normally distributed, maybe we care about outlier robustness, maybe our requirements demand to minimize the absolute error.

In this example, we will explore 6 flavors of linear regression: normal errors, double exponential (or Laplace) errors, and t-distributed errors. For each of these distribution, we will also study the effect of L^2 regularization, also known as Tikhonov regularization or ridge regression.

2.1.4 Double exponential errors

The double exponential distribution has the following PDF:

$$f(x \mid b, \mu) = \frac{1}{2b} \exp\{-\frac{1}{b}||x - \mu||_1\}$$

and if we write our linear regression as:

$$\boldsymbol{Y} \sim \text{DExp}(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 I)$$

it is easy to prove that finding the maximum likelihood estimator for $\boldsymbol{\beta}$ means minimizing the absolute (or L^1) error:

$$\min_{\boldsymbol{\beta}} ||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||_1 = \min_{\boldsymbol{\beta}} \sum_{i=1}^{n} |y_i - \boldsymbol{x_i}^T \boldsymbol{\beta}|$$

One of the strengths of this error distribution is its robustness with respect to the distribution of the response variable. This allows it to be used for modeling heavy tailed or asymmetric errors.

2.1.5 t-distributed errors

Another option for modeling the distribution of the errors is assuming that they follow a t-distribution. This is an additional method for handling heavy-tailed errors in which outliers might be present.

2.1.6 Ridge regression

Until this point, even though we are analyzing linear regression from a bayesian point of view, we have not specified a prior on the coefficients $\boldsymbol{\beta}$. We have derived the Maximum Likelihood Estimator for the coefficients, but doing so, it's as if we have implicitly used Maximum A Posteriori estimation with an improper uniform prior.

It turns out that if we decide to use normal prior on the coefficients, we automatically get what is called Ridge Regression. A type of regularized regression which is much more robust to colinearity in the explanatory variables.

Let's show how ridge regression naturally emerges from bayesian linear regression:

First of all let's define our priors and likelihood:

$$\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 I)$$
$$\boldsymbol{\beta} \sim \mathcal{N}(0, \lambda^{-1} I)$$

We want to find the mode of our posterior i.e. find a Maximum a Posteriori estimator for $\boldsymbol{\beta}$

$$\max_{\boldsymbol{\beta}} \pi(\boldsymbol{\beta}) \mathcal{L}(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\beta}) =$$

$$\max_{\boldsymbol{\beta}} \exp\{-\frac{1}{2\sigma^2}||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||^2\} \exp\{-\frac{\lambda}{2}||\boldsymbol{\beta}||^2\} =$$

$$\max_{\boldsymbol{\beta}} -\frac{1}{2\sigma^2}||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||^2 - \frac{\lambda}{2}||\boldsymbol{\beta}||^2 =$$

$$\min_{\boldsymbol{\beta}} \frac{1}{2\sigma^2}||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \frac{\lambda}{2}||\boldsymbol{\beta}||^2$$

This is the optimization problem we have to solve. In this form we see explicitly how the prior induces a regulation term on the coefficients, encouraging them to be close to zero.

This can be solved by setting the gradient to zero and using the Moore-Penrose pseudo-inverse.

$$\nabla_{\boldsymbol{\beta}} \frac{1}{2\sigma^2}||\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \frac{\lambda}{2}||\boldsymbol{\beta}||^2 = 0$$

$$\frac{1}{\sigma^2}\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta} = 0$$

We can remove the σ^2 factor by multiplying and keep the same λ with a small abuse of notation:

$$-\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta} = 0$$

$$-\boldsymbol{X}^T\boldsymbol{Y} + \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} = 0$$

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{Y}$$

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda I)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{Y}$$

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda I)^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$

2.2 Model

The JAGS model implements these the 6 regressions closely follows the formulas. The code shown below implements regression with normal errors and uninformative priors, while the other models are commented.

The first step consists in standardizing each column of the input variables.

Then the model is defined, connecting the observed variables Y with the mean of the predictors μ. In addition, for each observation, we are computing the residuals and classifying as outliers the points that are more than 2.5 standard deviations outside of the mean.

Finally we are using either uninformative normal priors for the betas or we are performing ridge regression by sampling the parameter φ (what we had called λ) from a Gamma distribution.

To sample from the posterior we use JAGS's MCMC with 2 chains, 100 adaptation rounds, 1000 burning rounds, and take 10000 samples per chain.

```
model {
  # Standardise x's and coefficients
  for (j in 1 : p) {
    b[j] <- beta[j] / sd(x[ , j ])
    for (i in 1 : N) {
      z[i, j] <- (x[i, j] - mean(x[, j])) / sd(x[ , j])
    }
  }

  b0 <- beta0 - b[1] * mean(x[, 1]) - b[2] * mean(x[, 2]) - b[3] * mean(x[, 3])

  # Model
  d <- 4; # degrees of freedom for t
  for (i in 1 : N) {
    Y[i] ~ dnorm(mu[i], tau)
    #      Y[i] ~ ddexp(mu[i], tau)
    #      Y[i] ~ dt(mu[i], tau, d)

    mu[i] <- beta0 + beta[1] * z[i, 1] + beta[2] * z[i, 2] + beta[3] * z[i, 3]
    stres[i] <- (Y[i] - mu[i]) / sigma
    outlier[i] <- step(stres[i] - 2.5) + step(-(stres[i] + 2.5) )
  }

  # Priors
  beta0 ~ dnorm(0, 0.00001)
  for (j in 1 : p) {
    beta[j] ~ dnorm(0, 0.00001)    # coeffs independent
    #      beta[j] ~ dnorm(0, phi) # coeffs exchangeable (ridge regression)
  }
  tau ~ dgamma(1.0E-3, 1.0E-3)
  #   phi ~ dgamma(1.0E-2,1.0E-2)
  # standard deviation of error distribution
  sigma <- sqrt(1 / tau) # normal errors
  #   sigma <- sqrt(2) / tau # double exponential errors
  #   sigma <- sqrt(d / (tau * (d - 2))); # t errors on d degrees of freedom
}
```

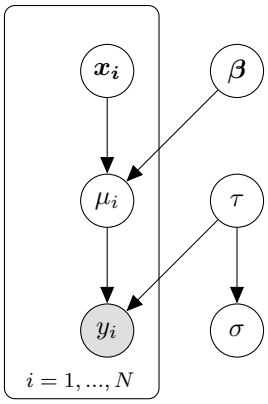

Figure 3: Graphical model for comparing linear regression models

2.3 Results

All six method give similar results for all parameters, confirming their consistency, as we can see from figure 4.

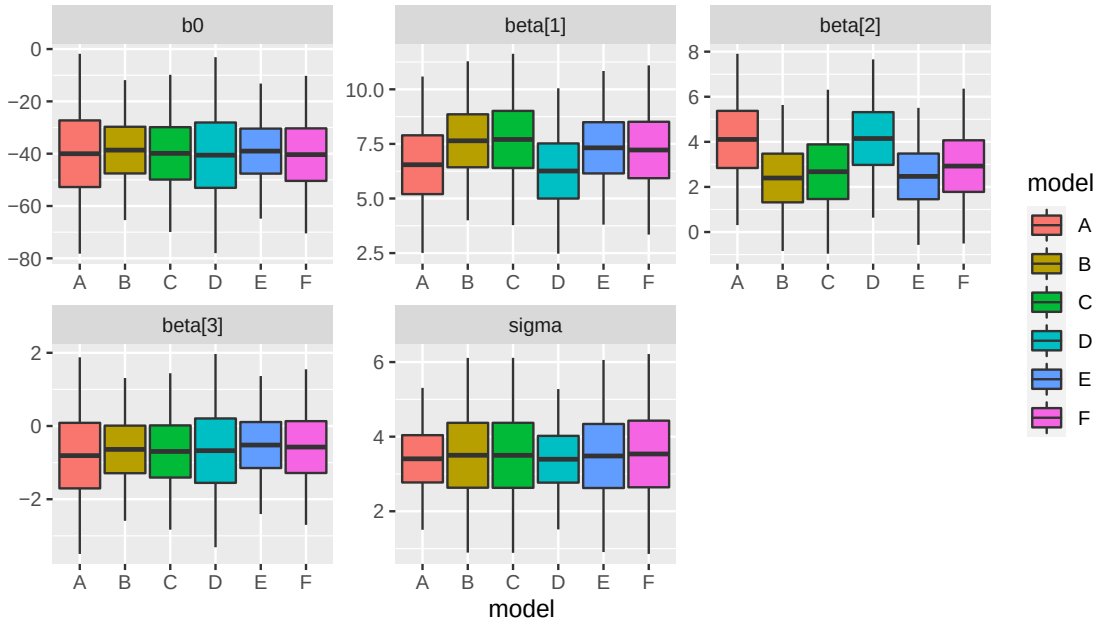

Figure 4: Comparing the distributions of parameters fitted by the 6 models. Model A: normal. Model B: double exponential. Model C: t-student. Model D: normal + Ridge. Model E: double exponential + Ridge. Model F: t-student + Ridge.

Additionaly, we can print all points that have been classified as outliers in each model. We can observe that the double exponential and t-student residual models classify more points as outliers. Finally, when adding L^2 regularization more points are classified as outliers in comparison to the unregularized model.

3 Alligators' food choice: working with categorical data

Let us now analyze a different problem, moving from linear regression with quantitative predictors to logistic regression with qualitative factors.

Our example comes from Introduction to Categorical Data Analysis, by Alan Agresti [1]. Agresti analyses a dataset on the alimentation of alligators. These alligators are classified by lake and by their length, separating them into ones longer than 2.3 meters and ones shorter than 2.3 meters. The primary food choices of the the alligators are five and we are interested in understanding how a specific food, lake, and length influence their distribution.

3.1 Theory

3.1.1 Logistic regression as probability

Since this example models the alligator's feeding choices with a multinomial random variable, let us begin by introducing a simpler case: logistic regression. We will present it in a probabilistic perspective as an instance of the Generalized Linear Model (GLM).

Each instance y_i of the dependent variable Y is modeled with a Bernoulli random variable.

$$y_i \sim \text{Bernoulli}(p)$$

And the probability of success p is linked to the explanatory variable through the log-odds:

$$\log \frac{p}{1-p} = \boldsymbol{X}\boldsymbol{\beta}$$

We can compute the reverse relationship, to obtain a function from the the input data to the probability:

$$p = \frac{1}{1 + e^{-\boldsymbol{X}\boldsymbol{\beta}}}$$

To obtain obtain an estimator for the coefficients we can again decide to use maximum likelihood estimation, which will yield a nonlinear system of equations. This system can be solved with iterative methods such as Newton Raphson .

3.1.2 Extensions of logistic regression

To model the alligator's data, two extensions to linear regression are required. For linear regression we are modeling each response as a Bernoulli trial with 2 possible outcomes.

We can extend it to multiple $K > 2$ classes by modeling the response using a categorical distribution (also called generalized Bernoulli distribution or multinulli distribution). This problem has now become multiclass logistic regression or softmax regression.

We can extend our model to multiple experiments $N > 1$ by treating the response as a binomial random variable, obtaining binomial regression.

And finally, when combining the two extensions (i.e. $K > 2$, $N > 1$) we obtain our model, where each response is modeled as a multinomial random variable.

3.1.3 Hierarchical models

An important feature of the Alligators example is that it is a hierarchical (or multilevel) model. A hierarchical model is a model with multiple levels, which allows parameters to be shared among different clusters in the observed data.

In our case we have dimensions onto which the data is distributed. There are differences between foods, differences between lakes, and difference between sizes. One approach to handle the case is to build multiple models for each class, referred to as pooling. But such an approach might not always produce reliable results. For example, the model corresponding to a class with very few students will be very misleading. A single

unpooled model might not be able to fit sufficiently on the data. We want to find a middle ground that finds a compromise between these extremes. This brings us to Bayesian hierarchical modeling.

3.2 Model

Below we show the JAGS code implementing the alligators hierarchical model .

The first step consists in specifying the priors for $\alpha_k, \beta_{ik}, \gamma_{jk}$. We are using normal uninformative prior with mean 0 and variance 10^5. The first element or column of the parameters is initialized to 0 in order to allow for easier comparison with the other parameters.

Then we define the likelihood, modeling each response X_{ij} as a multinomial random variable with n_{ij} (observed) experiments and a probability vector p_{ijk}. Each probability is defined as the sum of $\alpha_k, \beta_{ik}, \gamma_{jk}$. Since these sums are not guaranteed to be probabilities we apply for each food the softmax activation function to re-normalize them.

Finally we transform our outputs in order to compare them with Agresti's result and perform a G-test for goodness-of-fit. The G-statistic is proportional to $2N$ times the Kullback Leibler divergence.

To sample from the posterior we use JAGS's MCMC with 2 chains, 100 adaptation rounds, 1000 burning rounds, and take 10000 samples per chain.

```
var
 X[I,J,K],      # observations
 n[I,J],        # total for each covariate pattern
 E[I,J,K],      # fitted values
 OlogOE[I,J,K], # O log O/E
 G2,            # goodness-of-fit statistic
 mu[I,J,K],     # Poisson means
 phi[I,J,K],    # exp (beta[k] ' x[i,j])
 p[I,J,K],      # fitted probabilities
 lambda[I,J],   # baseline rates in each covariate strata
 alpha[K],      # factor for food = 2,3,4,5
 beta[I,K],     # factor for lakes = 2,3,4, for each food
 b[I,K],        # factor for lakes = 2,3,4, relative to food 1, centred
 gamma[J,K],    # factor for size = 2,  for each food
 g[J,K];        # factor for size = 2,  relative to food 1, centred
model {
   # PRIORS

   # Loop around foods
   alpha[1] <- 0;       # zero contrast for baseline food
   for (k in 2:K){
      alpha[k] ~ dnorm(0,0.00001);  # vague priors
   }

   # Loop around lakes:
   for (k in 1:K){
      beta[1,k] <- 0;   # corner-point contrast with first lake
   }
   for (i in 2:I) {
      beta[i,1] <- 0;   # zero contrast for baseline food
      for (k in 2:K) {
         beta[i,k] ~ dnorm(0,0.00001); # vague priors
      }
   }
```

```
   # Loop around sizes:
   for (k in 1:K){
      gamma[1,k] <- 0;  # corner-point contrast with first size
   }
   for (j in 2:J) {
      gamma[j,1] <- 0;  # zero contrast for baseline food
      for ( k in 2:K){
         gamma[j,k] ~ dnorm(0,0.00001); # vague priors
      }
   }

   # LIKELIHOOD
   for (i in 1:I) {        # loop around lakes
      for (j in 1:J) {     # loop around sizes
         # Multinomial response
         X[i,j,] ~ dmulti( p[i,j,] , n[i,j]  );
         for (k in 1:K) {  # loop around foods
            p[i,j,k]        <- phi[i,j,k] / sum(phi[i,j,]);
            log(phi[i,j,k]) <- alpha[k] + beta[i,k]  + gamma[j,k];
         }
      }
   }

   # TRANSFORM OUTPUT TO ENABLE COMPARISON WITH AGRESTI'S RESULTS
   for (k in 1:K) {        # loop around foods
      for (i in 1:I) {     # loop around lakes
         b[i,k] <- beta[i,k] - mean(beta[,k]);   # sum to zero constraint
      }
      for (j in 1:J) {     # loop around sizes
         g[j,k] <- gamma[j,k] - mean(gamma[,k]); # sum to zero constraint
      }
   }

   # FITTED VALUES
   for (i in 1:I) {     # loop around lakes
      for (j in 1:J) {     # loop around sizes
         for (k in 1:K) {     # loop around foods
            E[i,j,k] <- p[i,j,k] * n[i,j];
            OlogOE[i,j,k] <- X[i,j,k] * log( X[i,j,k] / E[i,j,k] );
         }
      }
   }
   G2 <- 2 * sum( OlogOE[,,] );
}
```

3.3 Results

We can analyze the results of the hierarchical model by box-plotting each one of the fitted parameters. In figure 6 we can visualize the values of α_k, i.e. the global popularity of every food. We can see that all foods appear to be less popular than the first food: fish.

Moving to the β parameter we can see in figure 7 that, by changing lake, the distribution of food choices changes. This could difference can be explained by noticing that each lake has a different habitat, thus a difference distribution of animals that the gators can decide to feed on.

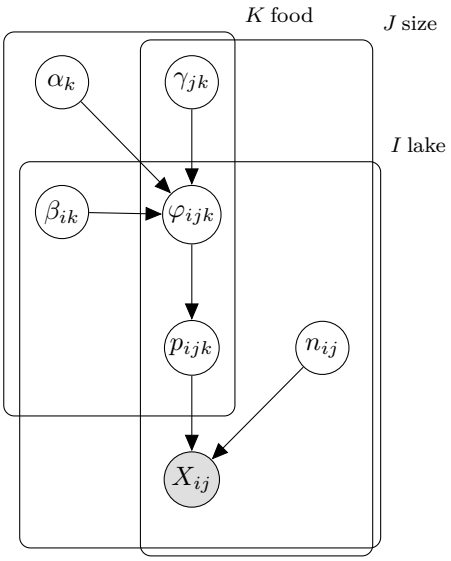

Figure 5: Graphical model for the second example: alligators

Finally we analyze the γ parameter which describes how size influences the feeding choices. We can see that large (size=2) alligators eat less invertebrates (food=2) and instead prefer reptiles or birds (foods=$3, 4$).

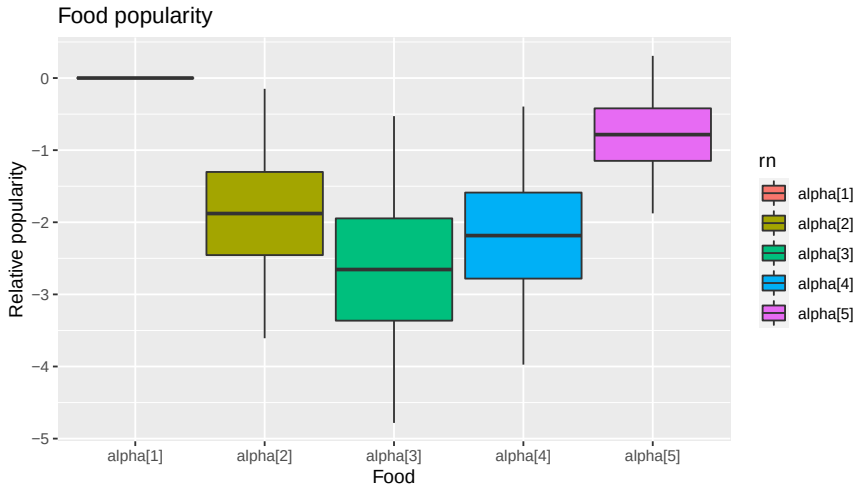

Figure 6: Food popularity

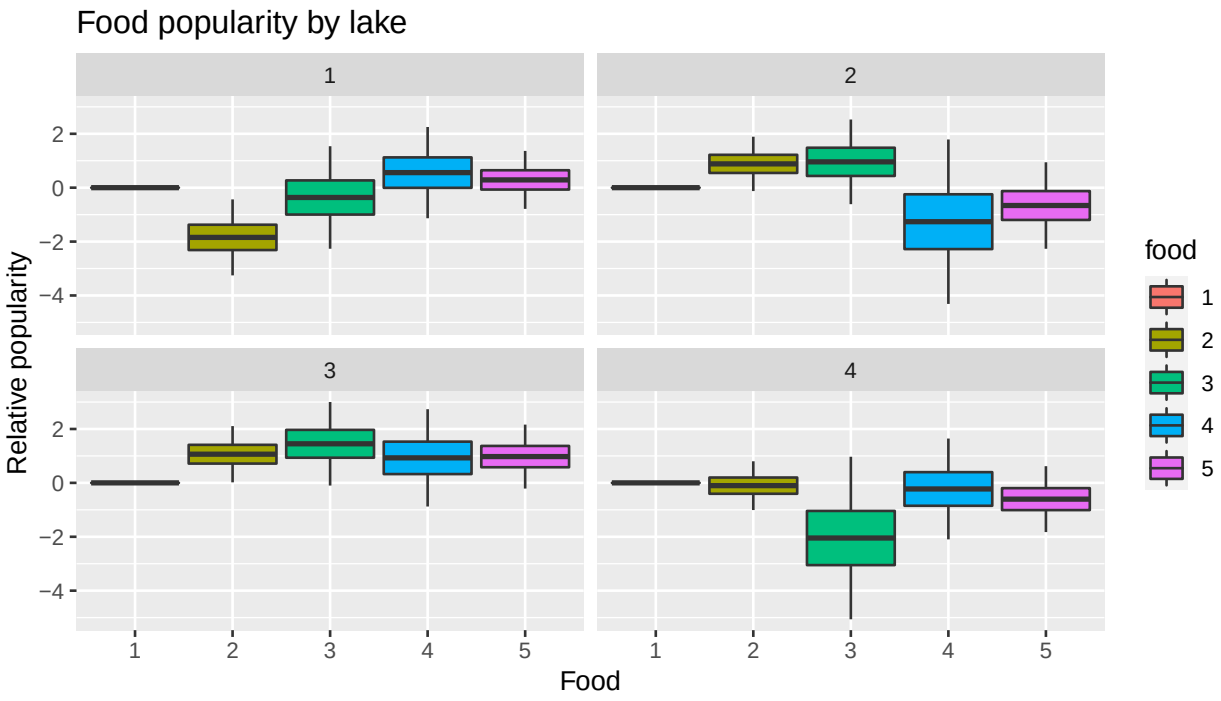

Figure 7: Food popularity by lake

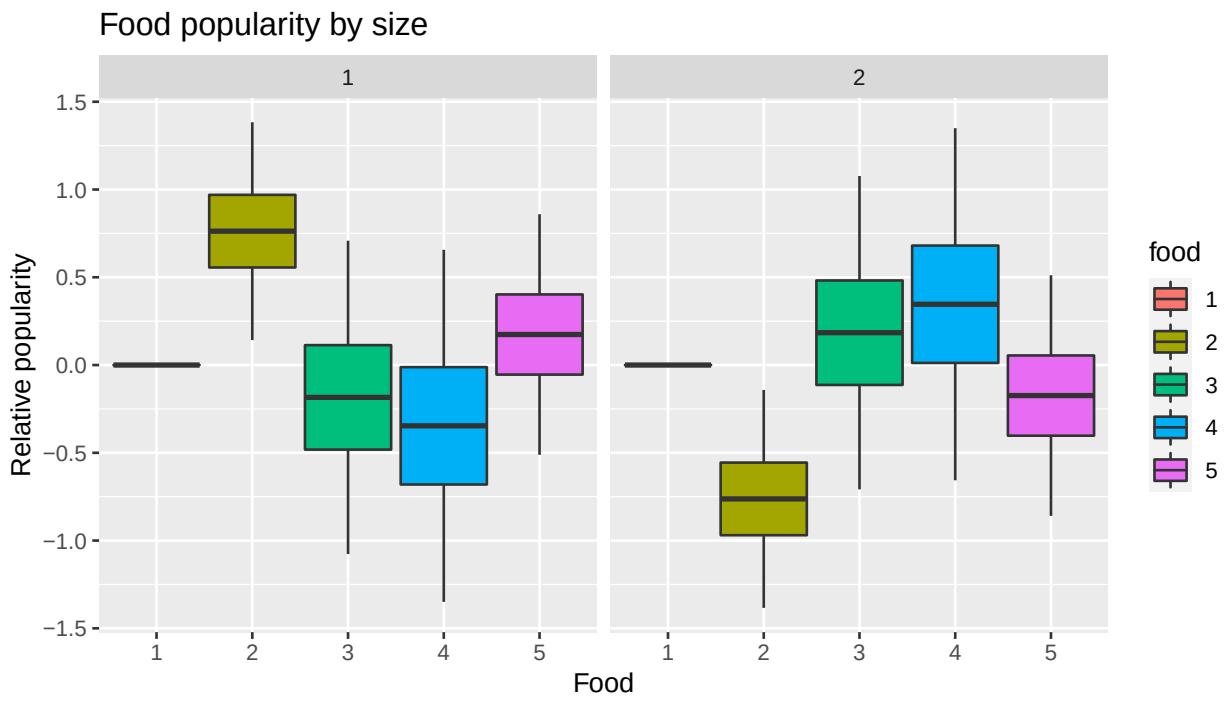

Figure 8: Food popularity by size

References

[1] Alan Agresti. *An introduction to categorical data analysis*. New York: Wiley, 1996, **pages** –. ISBN: 0471113387 9780471113386. URL: http://www.worldcat.org/search?qt=worldcat_org_all&q=0471113387.

[2] Yadolah Dodge David Birkes. *Alternative Methods of Regression*. Wiley, 1993.

[3] David J. Lunn **andothers**. "WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility". **in**: *Statistics and Computing* 10 (2000), **pages** 325–337.

[4] *Visual Explanation of Gibbs Sampling*. https://mikelove.wordpress.com/2008/09/08/visual-explanation-of-gibbs-sampling.