Lesson 9: Logistic regression

Lesson 9.2

Data

For an example of logistic regression, we’ll use the urine data set from the `boot` package in `R`. The response variable is `r`, which takes on values of 0 or 1. We will remove some rows from the data set which contain missing values.

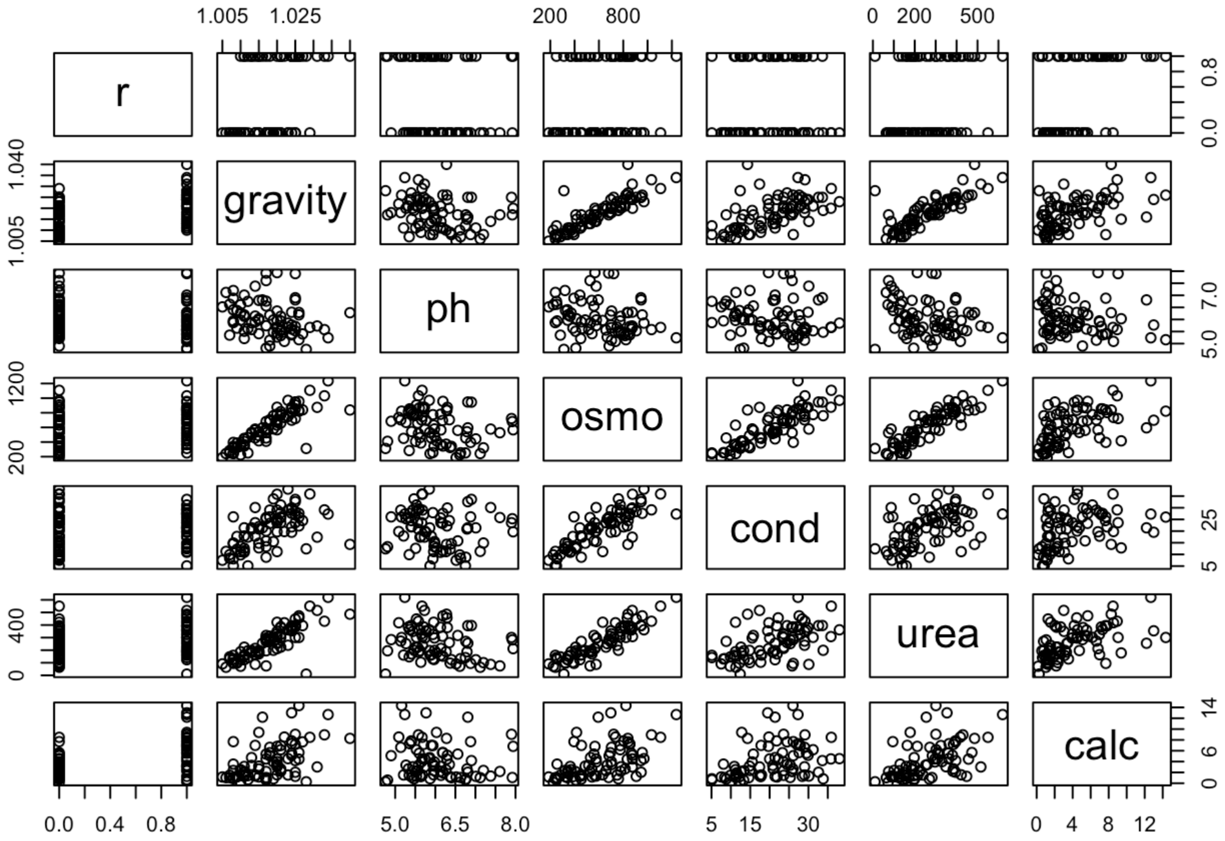
```
library("boot")
data("urine")
?urine
head(urine)
```

```
##   r gravity   ph osmo cond urea calc
## 1 0   1.021 4.91  725   NA  443 2.45
## 2 0   1.017 5.74  577 20.0  296 4.49
## 3 0   1.008 7.20  321 14.9  101 2.36
## 4 0   1.011 5.51  408 12.6  224 2.15
## 5 0   1.005 6.52  187  7.5   91 1.16
## 6 0   1.020 5.27  668 25.3  252 3.34
```

```
dat = na.omit(urine)
```

Let’s look at pairwise scatter plots of the seven variables.

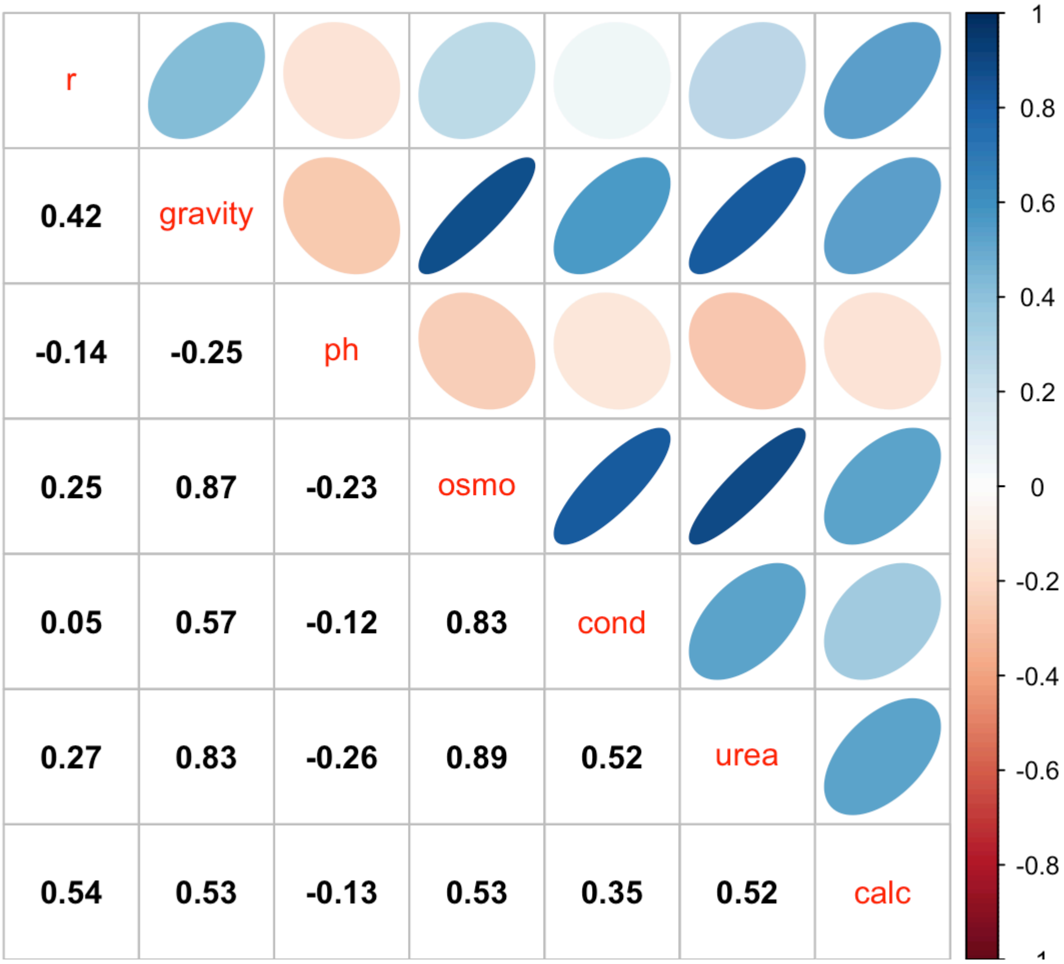
```
pairs(dat)
```



One thing that stands out is that several of these variables are strongly correlated with one another. For example `gravity` and `osmo` appear to have a very close linear relationship. Collinearity between x variables in linear regression models can cause trouble for statistical inference. Two correlated variables will compete for the ability to predict the response variable, leading to unstable estimates. This is not a problem for prediction of the response, if prediction is the end goal of the model. But if our objective is to discover *how* the variables relate to the response, we should avoid collinearity.

We can more formally estimate the correlation among these variables using the `corrplot` package.

```
library("corrplot")
Cor = cor(dat)
corrplot(Cor, type="upper", method="ellipse", tl.pos="d")
corrplot(Cor, type="lower", method="number", col="black",
         add=TRUE, diag=FALSE, tl.pos="n", cl.pos="n")
```



Variable selection

One primary goal of this analysis is to find out which variables are related to the presence of calcium oxalate crystals. This objective is often called “variable selection.” We have already seen one way to do this: fit several models that include different sets of variables and see which one has the best DIC. Another way to do this is to use a linear model where the priors for the β coefficients favor values near 0 (indicating a weak relationship). This way, the burden of establishing association lies with the data. If there is not a strong signal, we assume it doesn’t exist.

Rather than tailoring a prior for each individual β based on the scale its covariate takes values on, it is customary to subtract the mean and divide by the standard deviation for each variable.

```
X = scale(dat[,-1], center=TRUE, scale=TRUE)
head(X[,"gravity"])
```

```
##          2          3          4          5          6          7 
## -0.1403037 -1.3710690 -0.9608139 -1.7813240  0.2699514 -0.8240622
```

```
colMeans(X)
```

```
##       gravity            ph          osmo          cond          urea 
## -9.861143e-15  8.511409e-17  1.515743e-16 -1.829852e-16  7.335402e-17 
##          calc 
## -1.689666e-18
```

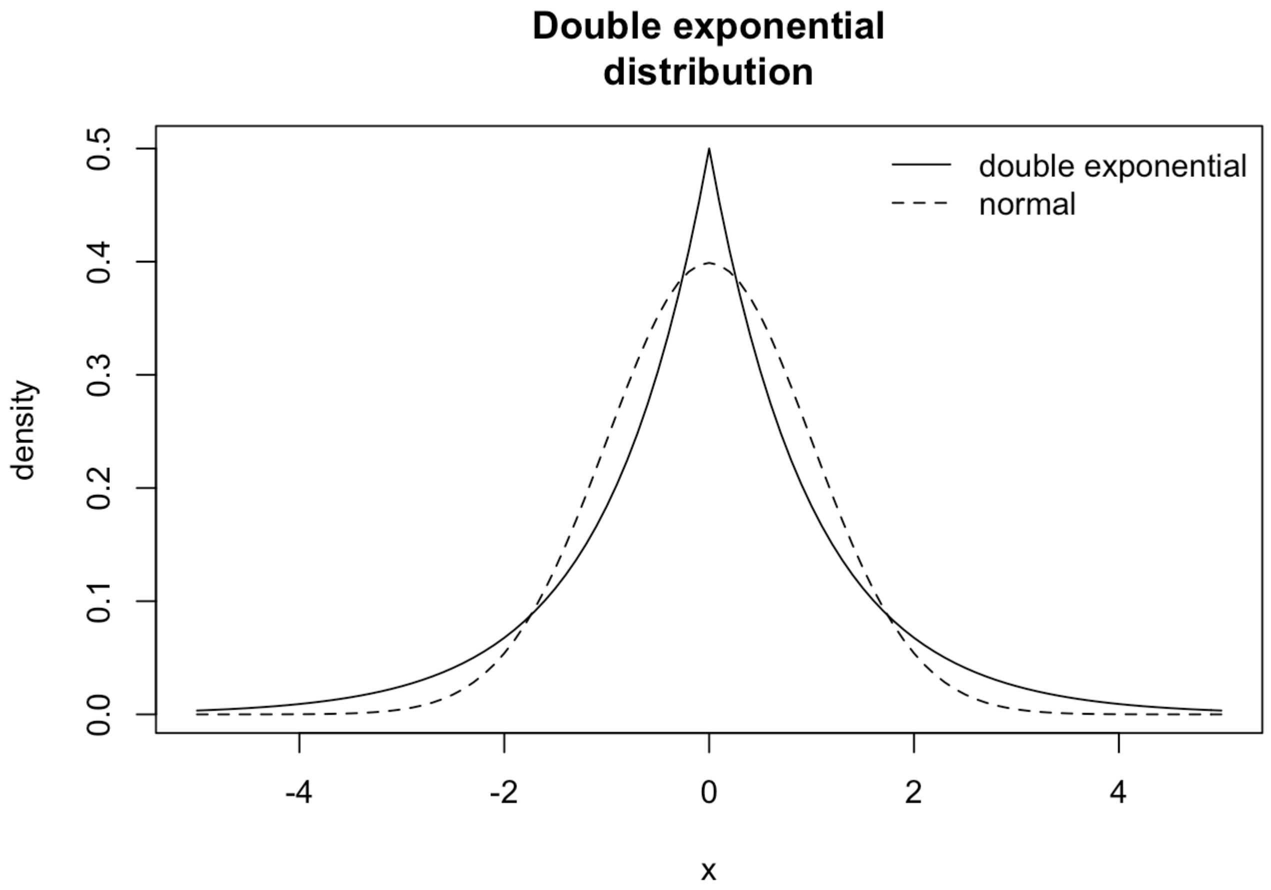
```
apply(X, 2, sd)
```

```
## gravity      ph    osmo    cond    urea    calc
##       1       1       1       1       1       1
```

Model

Our prior for the β (which we'll call b in the model) coefficients will be the double exponential (or Laplace) distribution, which as the name implies, is the exponential distribution with tails extending in the positive direction as well as the negative direction, with a sharp peak at 0. We can read more about it in the `JAGS` manual. The distribution looks like:

```
ddexp = function(x, mu, tau) {
  0.5*tau*exp(-tau*abs(x-mu))
}
curve(ddexp(x, mu=0.0, tau=1.0), from=-5.0, to=5.0, ylab="density", main="Double exponential\ndistribution") # double exponential distribution
curve(dnorm(x, mean=0.0, sd=1.0), from=-5.0, to=5.0, lty=2, add=TRUE) # normal distribution
legend("topright", legend=c("double exponential", "normal"), lty=c(1,2), bty="n")
```



```
library("rjags")
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.2.0
## Loaded modules: basemod,bugs
```

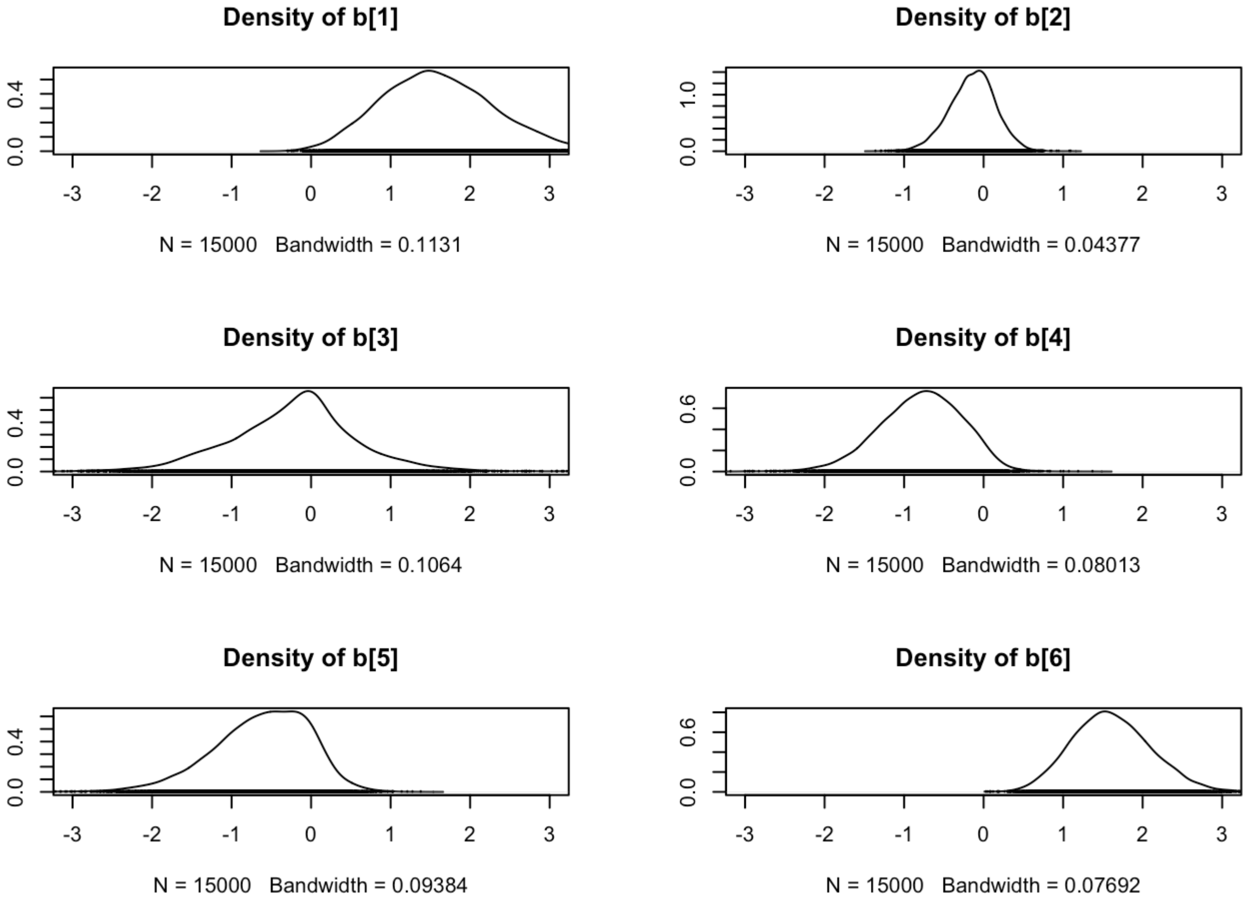
```
mod1_string = " model {
    for (i in 1:length(y)) {
        y[i] ~ dbern(p[i])
        logit(p[i]) = int + b[1]*gravity[i] + b[2]*ph[i] + b[3]*osmo[i] + b[4]*cond
[i] + b[5]*urea[i] + b[6]*calc[i]
    }
    int ~ dnorm(0.0, 1.0/25.0)
    for (j in 1:6) {
        b[j] ~ ddexp(0.0, sqrt(2.0)) # has variance 1.0
    }
} "

set.seed(92)
head(X)

data_jags = list(y=dat$r, gravity=X[,"gravity"], ph=X[,"ph"], osmo=X[,"osmo"], cond=X
[,"cond"], urea=X[,"urea"], calc=X[,"calc"])

params = c("int", "b")

mod1 = jags.model(textConnection(mod1_string), data=data_jags, n.chains=3)
update(mod1, 1e3)

mod1_sim = coda.samples(model=mod1,
                        variable.names=params,
                        n.iter=5e3)
mod1_csim = as.mcmc(do.call(rbind, mod1_sim))

## convergence diagnostics
plot(mod1_sim, ask=TRUE)

gelman.diag(mod1_sim)
autocorr.diag(mod1_sim)
autocorr.plot(mod1_sim)
effectiveSize(mod1_sim)

## calculate DIC
dic1 = dic.samples(mod1, n.iter=1e3)
```

Let’s look at the results.

```
summary(mod1_sim)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean     SD Naive SE Time-series SE
## b[1]  1.6202 0.7303 0.005963       0.019347
## b[2] -0.1393 0.2870 0.002344       0.003150
## b[3] -0.2243 0.8036 0.006561       0.028292
## b[4] -0.7862 0.5172 0.004223       0.013882
## b[5] -0.6240 0.6057 0.004946       0.016679
## b[6]  1.6123 0.4965 0.004054       0.006862
## int  -0.1787 0.3086 0.002520       0.003568
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%      75%  97.5%
## b[1]  0.3386  1.1024  1.5701  2.08242 3.1819
## b[2] -0.7332 -0.3255 -0.1245  0.05310 0.4018
## b[3] -1.8648 -0.6967 -0.1722  0.22356 1.3878
## b[4] -1.8574 -1.1249 -0.7618 -0.41734 0.1221
## b[5] -1.9618 -1.0037 -0.5622 -0.17451 0.3751
## b[6]  0.7112  1.2643  1.5859  1.93499 2.6432
## int  -0.7736 -0.3866 -0.1807  0.02542 0.4404
```

```
par(mfrow=c(3,2))
densplot(mod1_csim[,1:6], xlim=c(-3.0, 3.0))
```



```
colnames(X) # variable names
```

```
## [1] "gravity" "ph"      "osmo"    "cond"    "urea"    "calc"
```

It is clear that the coefficients for variables `gravity`, `cond` (conductivity), and `calc` (calcium concentration) are not 0. The posterior distribution for the coefficient of `osmo` (osmolarity) looks like the prior, and is almost centered on 0 still, so we'll conclude that `osmo` is not a strong predictor of calcium oxalate crystals. The same goes for `ph`.

`urea` (urea concentration) appears to be a borderline case. However, if we refer back to our correlations among the variables, we see that `urea` is highly correlated with `gravity`, so we opt to remove it.

Our second model looks like this:

```
mod2_string = " model {
    for (i in 1:length(y)) {
        y[i] ~ dbern(p[i])
        logit(p[i]) = int + b[1]*gravity[i] + b[2]*cond[i] + b[3]*calc[i]
    }
    int ~ dnorm(0.0, 1.0/25.0)
    for (j in 1:3) {
        b[j] ~ dnorm(0.0, 1.0/25.0) # noninformative for logistic regression
    }
} "

mod2 = jags.model(textConnection(mod2_string), data=data_jags, n.chains=3)
```

```
## Warning in jags.model(textConnection(mod2_string), data = data_jags,
## n.chains = 3): Unused variable "ph" in data
```

```
## Warning in jags.model(textConnection(mod2_string), data = data_jags,
## n.chains = 3): Unused variable "osmo" in data
```

```
## Warning in jags.model(textConnection(mod2_string), data = data_jags,
## n.chains = 3): Unused variable "urea" in data
```

```
update(mod2, 1e3)

mod2_sim = coda.samples(model=mod2,
                        variable.names=params,
                        n.iter=5e3)
mod2_csim = as.mcmc(do.call(rbind, mod2_sim))

plot(mod2_sim, ask=TRUE)

gelman.diag(mod2_sim)
autocorr.diag(mod2_sim)
autocorr.plot(mod2_sim)
effectiveSize(mod2_sim)

dic2 = dic.samples(mod2, n.iter=1e3)
```

Results

```
dic1
```

```
## Mean deviance:  68.56
## penalty 5.328
## Penalized deviance: 73.88
```

```
dic2
```

```
## Mean deviance:  71.13
## penalty 4.007
## Penalized deviance: 75.13
```

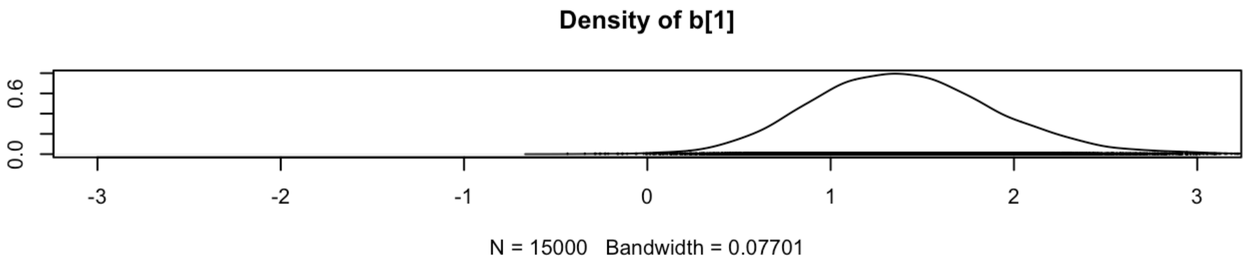
```
summary(mod2_sim)
```

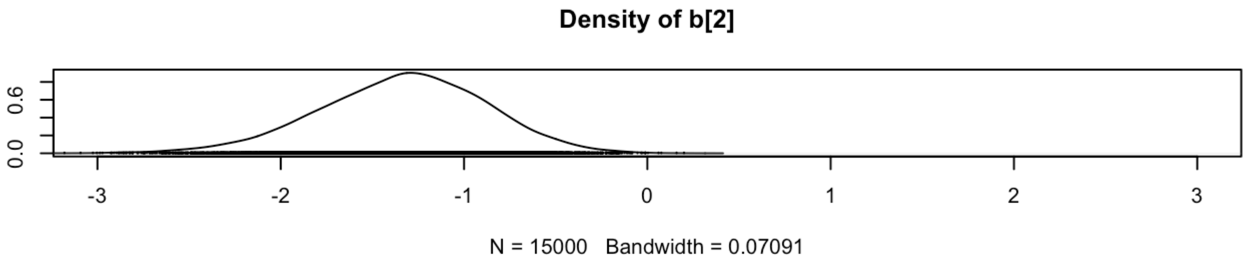
```
## 
## Iterations = 2001:7000
## Thinning interval = 1 
## Number of chains = 3 
## Sample size per chain = 5000 
## 
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
## 
##         Mean     SD Naive SE Time-series SE
## b[1]  1.4033 0.5049 0.004123       0.008597
## b[2] -1.3366 0.4577 0.003737       0.008707
## b[3]  1.8762 0.5431 0.004435       0.008457
## int  -0.1476 0.3228 0.002635       0.003741
## 
## 2. Quantiles for each variable:
## 
##         2.5%     25%     50%      75%   97.5%
## b[1]  0.4876  1.0550  1.3784  1.72110  2.4576
## b[2] -2.2938 -1.6337 -1.3159 -1.02039 -0.4888
## b[3]  0.8989  1.4959  1.8460  2.21792  3.0266
## int  -0.7786 -0.3652 -0.1498  0.06965  0.4955
```

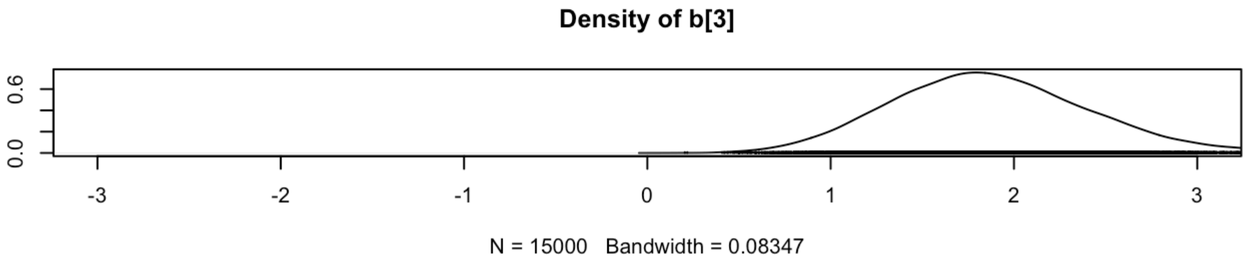
```
HPDinterval(mod2_csim)
```

```
##           lower      upper
## b[1]  0.4368820  2.3957538
## b[2] -2.2578998 -0.4619907
## b[3]  0.8613524  2.9670466
## int  -0.7889525  0.4837506
## attr(,"Probability")
## [1] 0.95
```

```
par(mfrow=c(3,1))
densplot(mod2_csim[,1:3], xlim=c(-3.0, 3.0))
```







```
colnames(X)[c(1,4,6)] # variable names
```

```
## [1] "gravity" "cond"    "calc"
```

The DIC is actually better for the first model. Note that we did change the prior between models, and generally we should not use the DIC to choose between priors. Hence comparing DIC between these two models may not be a fair comparison. Nevertheless, they both yield essentially the same conclusions. Higher values of `gravity` and `calc` (calcium concentration) are associated with higher probabilities of calcium oxalate crystals, while higher values of `cond` (conductivity) are associated with lower probabilities of calcium oxalate crystals.

There are more modeling options in this scenario, perhaps including transformations of variables, different priors, and interactions between the predictors, but we'll leave it to you to see if you can improve the model.

Lesson 9.3

Prediction from a logisic regression model

How do we turn model parameter estimates into model predictions? The key is the form of the model. Remember that the likelihood is Bernoulli, which is 1 with probability p. We modeled the logit of p as a linear model, which we showed in the first segment of this lesson leads to an exponential form for E$(y) = p$.

Take the output from our model in the last segment. We will use the posterior means as point estimates of the parameters.

```
(pm_coef = colMeans(mod2_csim))
```

```
##       b[1]       b[2]       b[3]        int
##  1.4032534 -1.3366435  1.8762089 -0.1476021
```

The posterior mean of the intercept was about -0.15. Since we centered and scaled all of the covariates, values of 0 for each x correspond to the average values. Therefore, if we use our last model, then our point estimate for the probability of calcium oxalate crystals when `gravity` , `cond` , and `calc` are at their average values is $1/(1+e^{-(-0.15)}) =$ 0.4625702.

Now suppose we want to make a prediction for a new specimen whose value of `gravity` is average, whose value of `cond` is one standard deviation below the mean, and whose value of `calc` is one standard deviation above the mean. Our point estimate for the probability of calcium oxalate crystals is $1/(1+e^{-(-0.15+1.4*0.0-1.3*(-1.0)+1.9*(1.0))}) =$ 0.9547825.

If we want to make predictions in terms of the original x variable values, we have two options:

1. For each x variable, subtract the mean and divide by the standard deviation for that variable in the original data set used to fit the model.

2. Re-fit the model without centering and scaling the covariates.

Predictive checks

We can use the same ideas to make predictions for each of the original data points. This is similar to what we did to calculate residuals with earlier models.

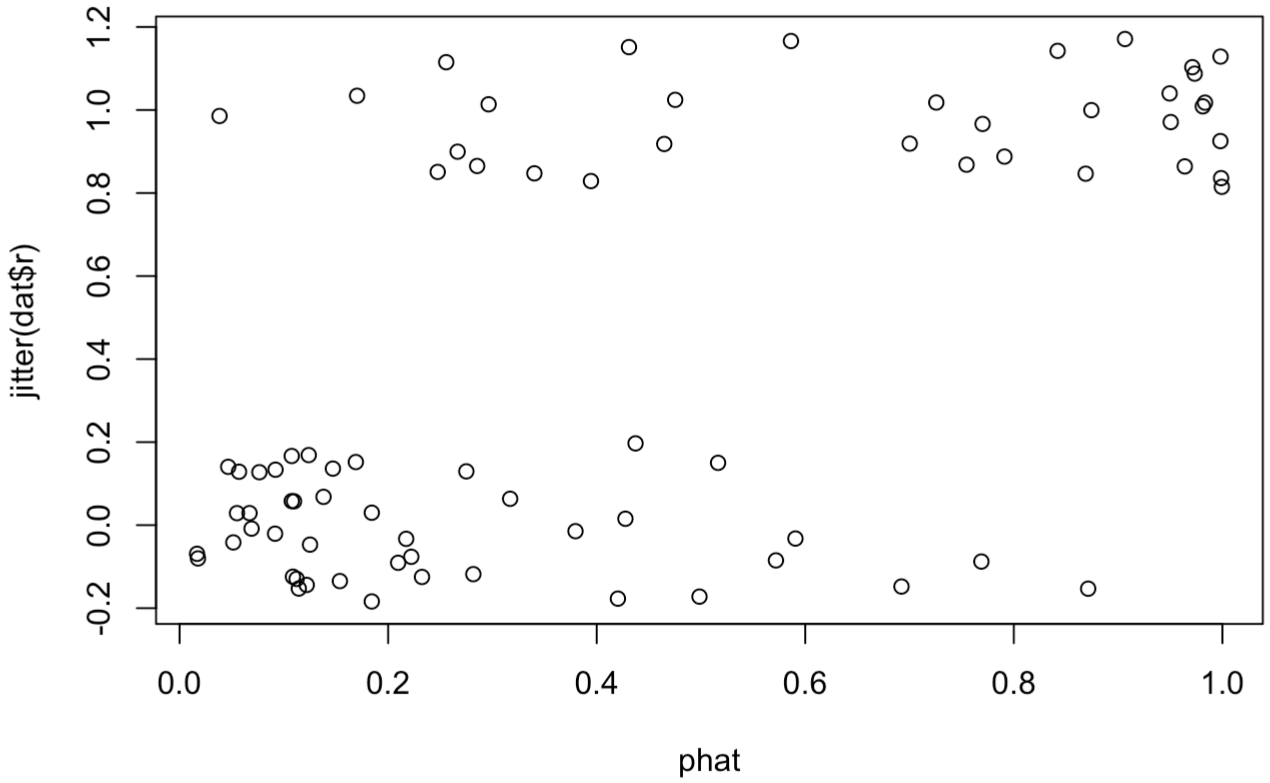
First we take the X matrix and matrix multiply it with the posterior means of the coefficients. Then we need to pass these linear values through the inverse of the link function as we did above.

```
pm_Xb = pm_coef["int"] + X[,c(1,4,6)] %*% pm_coef[1:3]
phat = 1.0 / (1.0 + exp(-pm_Xb))
head(phat)
```

```
##         [,1]
## 2 0.49856339
## 3 0.10975009
## 4 0.22217686
## 5 0.10753888
## 6 0.27496692
## 7 0.09199544
```

These `phat` values are the model’s predicted probability of calcium oxalate crystals for each data point. We can get a rough idea of how successful the model is by plotting these predicted values against the actual outcome.

```
plot(phat, jitter(dat$r))
```



Suppose we choose a cutoff for these predicted probabilities. If the model tells us the probability is higher than 0.5, we will classify the observation as a 1 and if it is less than 0.5, we will classify it as a 0. That way the model classifies each data point. Now we can tabulate these classifications against the truth to see how well the model predicts the original data.

```
(tab0.5 = table(phat > 0.5, data_jags$y))
```

```
## 
##          0  1
##   FALSE 38 12
##   TRUE   6 21
```

```
sum(diag(tab0.5)) / sum(tab0.5)
```

```
## [1] 0.7662338
```

The correct classification rate is about 76%, not too bad, but not great.

Now suppose that it is considered really bad to predict no calcium oxalate crystal when there in fact is one. We might then choose to lower our threshold for classifying data points as 1s. Say we change it to 0.3. That is, if the model says the probability is greater than 0.3, we will classify it as having a calcium oxalate crystal.

```
(tab0.3 = table(phat > 0.3, data_jags$y))
```

```
##
##          0  1
##   FALSE 32  7
##   TRUE  12 26
```

```
sum(diag(tab0.3)) / sum(tab0.3)
```

```
## [1] 0.7532468
```

It looks like we gave up a little classification accuracy, but we did indeed increase our chances of detecting a true positive.

We could repeat this exercise for many thresholds between 0 and 1, and each time calculate our error rates. This is equivalent to calculating what is called the ROC (receiver-operating characteristic) curve, which is often used to evaluate classification techniques.

These classification tables we have calculated were all in-sample. They were predicting for the same data used to fit the model. We could get a less biased assessment of how well our model performs if we calculated these tables for data that were not used to fit the model. For example, before fitting the model, you could withhold a set of randomly selected “test” data points, and use the model fit to the rest of the “training” data to make predictions on your “test” set.