17.8: Bayesian Regression

Okay, so now we've seen Bayesian equivalents to orthodox chi-square tests and t-tests. What's next? If I were to follow the same progression that I used when developing the orthodox tests you'd expect to see ANOVA next, but I think it's a little clearer if we start with regression.

17.8.1 quick refresher

In Chapter 15 I used the `parenthood` data to illustrate the basic ideas behind regression. To remind you of what that data set looks like, here's the first six observations:

```
load("./rbook-master/data/parenthood.Rdata")
head(parenthood)
```

```
##   dan.sleep baby.sleep dan.grump day
## 1      7.59      10.18        56   1
## 2      7.91      11.66        60   2
## 3      5.14       7.92        82   3
## 4      7.71       9.61        55   4
## 5      6.68       9.75        67   5
## 6      5.99       5.04        72   6
```

Back in Chapter 15 I proposed a theory in which my grumpiness (`dan.grump`) on any given day is related to the amount of sleep I got the night before (`dan.sleep`), and possibly to the amount of sleep our baby got (`baby.sleep`), though probably not to the `day` on which we took the measurement. We tested this using a regression model. In order to estimate the regression model we used the `lm()` function, like so:

```
model <- lm(
  formula = dan.grump ~ dan.sleep + day + baby.sleep,
  data = parenthood
)
```

The hypothesis tests for each of the terms in the regression model were extracted using the `summary()` function as shown below:

```
summary(model)
```

```
##
## Call:
## lm(formula = dan.grump ~ dan.sleep + day + baby.sleep, data = parenthood)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.906  -2.284  -0.295   2.652  11.880
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 126.278707   3.242492  38.945   <2e-16 ***
## dan.sleep    -8.969319   0.560007 -16.016   <2e-16 ***
## day          -0.004403   0.015262  -0.288    0.774
## baby.sleep    0.015747   0.272955   0.058    0.954
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.375 on 96 degrees of freedom
## Multiple R-squared:  0.8163, Adjusted R-squared:  0.8105
## F-statistic: 142.2 on 3 and 96 DF,  p-value: < 2.2e-16
```

When interpreting the results, each row in this table corresponds to one of the possible predictors. The `(Intercept)` term isn't usually interesting, though it is highly significant. The important thing for our purposes is the fact that `dan.sleep` is significant at p<.001 and neither of the other variables are.

17.8.2 Bayesian version

Okay, so how do we do the same thing using the `BayesFactor` package? The easiest way is to use the `regressionBF()` function instead of `lm()`. As before, we use `formula` to indicate what the full regression model looks like, and the `data` argument to specify the data frame. So the command is:

```
regressionBF(
  formula = dan.grump ~ dan.sleep + day + baby.sleep,
  data = parenthood
)
```

```
## Bayes factor analysis
## --------------
## [1] dan.sleep                   : 1.622545e+34 ±0.01%
## [2] day                         : 0.2724027    ±0%
## [3] baby.sleep                  : 10018411     ±0%
## [4] dan.sleep + day             : 1.016576e+33 ±0%
## [5] dan.sleep + baby.sleep      : 9.77022e+32  ±0%
## [6] day + baby.sleep            : 2340755      ±0%
## [7] dan.sleep + day + baby.sleep : 7.835625e+31 ±0%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

So that's pretty straightforward: it's exactly what we've been doing throughout the book. The output, however, is a little different from what you get from `lm()`. The format of this is pretty familiar. At the bottom we have some techical rubbish, and at the top we have some information about the Bayes factors. What's new is the fact that we seem to have *lots* of Bayes factors here. What's all this about?

The trick to understanding this output is to recognise that if we're interested in working out which of the 3 predictor variables are related to `dan.grump`, there are actually 8 possible regression models that could be considered. One possibility is the *intercept only model*, in which none of the three variables have an effect. At the other end of the spectrum is the *full model* in which all three variables matter. So what `regressionBF()` does is treat the *intercept only* model as the null hypothesis, and print out the Bayes factors for all other models when compared against that null. For example, if we look at line 4 in the table, we see that the evidence is about 1033 to 1 in favour of the claim that a model that includes both `dan.sleep` and `day` is better than the intercept only model. Or if we look at line 1, we can see that the odds are about 1.6×1034 that a model containing the `dan.sleep` variable (but no others) is better than the intercept only model.

17.8.3 Finding the best model

In practice, this isn't super helpful. In most situations the intercept only model is one that you don't really care about at all. What I find helpful is to start out by working out which model is the *best* one, and then seeing how well all the alternatives compare to it. Here's how you do that. In this case, it's easy enough to see that the best model is actually the one that contains `dan.sleep` only (line 1), because it has the largest Bayes factor. However, if you've got a lot of possible models in the output, it's handy to know that you can use the `head()` function to pick out the best few models. First, we have to go back and save the Bayes factor information to a variable:

```
models <- regressionBF(
  formula = dan.grump ~ dan.sleep + day + baby.sleep,
  data = parenthood
)
```

Let's say I want to see the best three models. To do this, I use the `head()` function specifying `n=3`, and here's what I get as the result:

```
head( models, n = 3)
```

```
## Bayes factor analysis
## --------------
## [1] dan.sleep              : 1.622545e+34 ±0.01%
## [2] dan.sleep + day        : 1.016576e+33 ±0%
## [3] dan.sleep + baby.sleep : 9.77022e+32  ±0%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

This is telling us that the model in line 1 (i.e., `dan.grump ~ dan.sleep`) is the best one. That's *almost* what I'm looking for, but it's still comparing all the models against the intercept only model. That seems silly. What I'd like to know is how big the difference is between the best model and the other good models. For that, there's this trick:

```
head( models/max(models), n = 3)
```

```
## Bayes factor analysis
## --------------
## [1] dan.sleep              : 1         ±0%
## [2] dan.sleep + day        : 0.0626532 ±0.01%
## [3] dan.sleep + baby.sleep : 0.0602154 ±0.01%
##
## Against denominator:
##   dan.grump ~ dan.sleep
## ---
## Bayes factor type: BFlinearModel, JZS
```

Notice the bit at the bottom showing that the "denominator" has changed. What that means is that the Bayes factors are now comparing each of those 3 models listed against the `dan.grump ~ dan.sleep` model. Obviously, the Bayes factor in the first line is exactly 1, since that's just comparing the best model to itself. More to the point, the other two Bayes factors are both less than 1, indicating that they're all worse than that model. The Bayes factors of 0.06 to 1 imply that the odds for the best model over the second best model are about 16:1. You can work this out by simple arithmetic (i.e., $0.06/1 \approx 16$), but the other way to do it is to directly compare the models. To see what I mean, here's the original output:

```
models
```

```
## Bayes factor analysis
## --------------
## [1] dan.sleep                   : 1.622545e+34 ±0.01%
## [2] day                         : 0.2724027    ±0%
## [3] baby.sleep                  : 10018411     ±0%
## [4] dan.sleep + day             : 1.016576e+33 ±0%
## [5] dan.sleep + baby.sleep      : 9.77022e+32  ±0%
## [6] day + baby.sleep            : 2340755      ±0%
## [7] dan.sleep + day + baby.sleep : 7.835625e+31 ±0%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

The best model corresponds to row 1 in this table, and the second best model corresponds to row 4. All you have to do to compare these two models is this:

```
models[1] / models[4]
```

```
## Bayes factor analysis
## --------------
## [1] dan.sleep : 15.96088 ±0.01%
##
## Against denominator:
##   dan.grump ~ dan.sleep + day
## ---
## Bayes factor type: BFlinearModel, JZS
```

And there you have it. You've found the regression model with the highest Bayes factor (i.e., `dan.grump ~ dan.sleep`), and you know that the evidence for that model over the next best alternative (i.e., `dan.grump ~ dan.sleep + day`) is about 16:1.

17.8.4 Extracting Bayes factors for all included terms

Okay, let's say you've settled on a specific regression model. What Bayes factors should you report? In this example, I'm going to pretend that you decided that `dan.grump ~ dan.sleep + baby.sleep` is the model you think is best. Sometimes it's sensible to do this, even when it's not the one with the highest Bayes factor. Usually this happens because you have a substantive theoretical reason to prefer one model over the other. However, in this case I'm doing it because I want to use a model with more than one predictor as my example!

Having figured out which model you prefer, it can be really useful to call the `regressionBF()` function and specifying `whichModels="top"`. You use your "preferred" model as the `formula` argument, and then the output will show you the Bayes factors that result when you try to drop predictors from this model:

```
regressionBF(
 formula = dan.grump ~ dan.sleep + baby.sleep,
 data = parenthood,
 whichModels = "top"
)
```

```
## Bayes factor top-down analysis
## --------------
## When effect is omitted from dan.sleep + baby.sleep , BF is...
## [1] Omit baby.sleep : 16.60705     ±0.01%
## [2] Omit dan.sleep  : 1.025403e-26 ±0.01%
##
## Against denominator:
##   dan.grump ~ dan.sleep + baby.sleep
## ---
## Bayes factor type: BFlinearModel, JZS
```

Okay, so now you can see the results a bit more clearly. The Bayes factor when you try to drop the `dan.sleep` predictor is about 10^{-26}, which is very strong evidence that you *shouldn't* drop it. On the other hand, the Bayes factor actually goes up to 17 if you drop `baby.sleep`, so you'd usually say that's pretty strong evidence for dropping that one.
