Recap: prediction

Olivia Burge

6 October 2014

This handout:

(1) provides a worked example of predicting from a poisson glm;
(2) explains how to use the `effects` package to plot the results similarly;
(3) explains how to use and install the `coefplot2` package to visualise the *coefficients* returned by `summary(model)` where model is the name of the model; and
(4) demonstrates why you may wish to choose (1) above rather than (2) for more complex analyses.

(1) Worked prediction example

Data

This data is from the vegan dataset. You'll need to have loaded it to access it.

```
data(mite)
data(mite.env)

#see an explanation of any dataset that comes with R:
?mite

#see the first couple of rows
head(mite[1:10], 3) # and only first 10 columns.  All the columns are species.
```

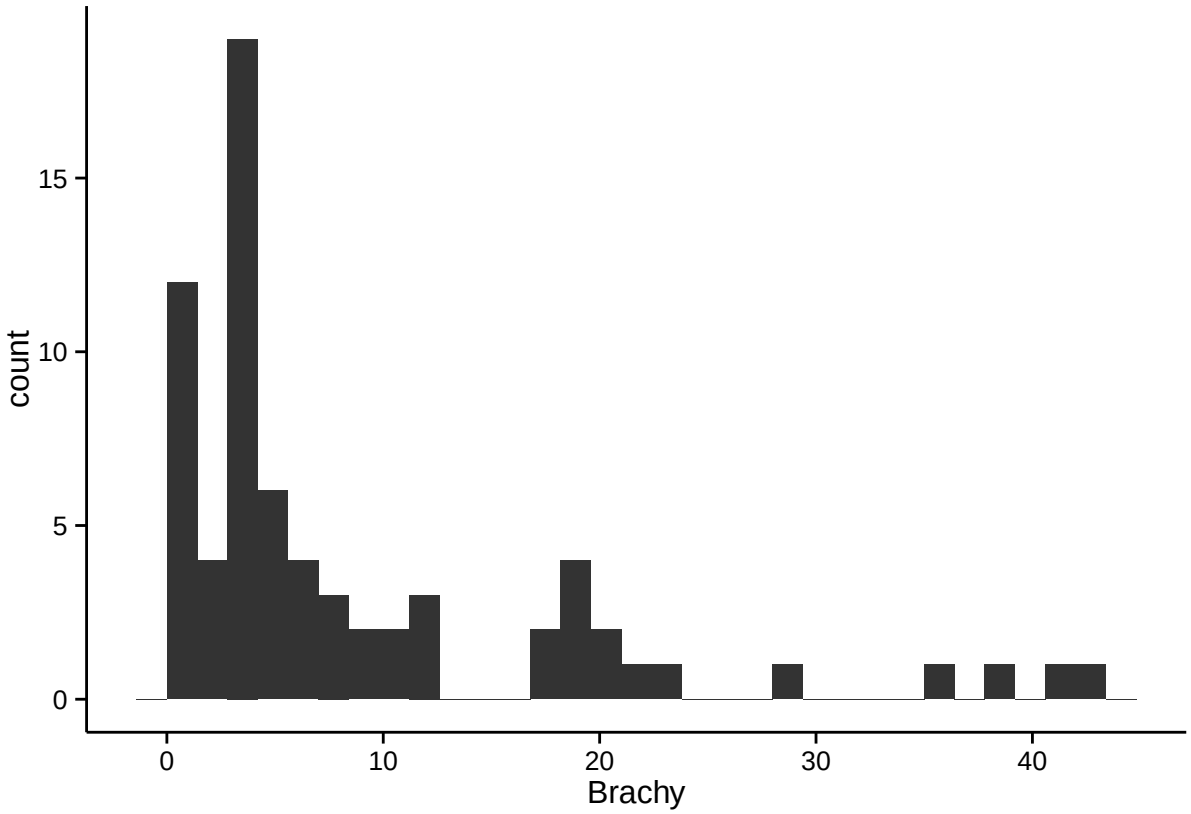
```
##   Brachy PHTH HPAV RARD SSTR Protopl MEGR MPRO TVIE HMIN
## 1     17    5    5    3    2       1    4    2    2    1
## 2      2    7   16    0    6       0    4    2    0    0
## 3      4    3    1    1    2       0    3    0    0    0
```

```
head(mite.env, 3)
```

```
##   SubsDens WatrCont Substrate Shrub    Topo
## 1    39.18    350.1   Sphagn1   Few Hummock
## 2    54.99    434.8    Litter   Few Hummock
## 3    46.07    371.7 Interface   Few Hummock
```

```
#quick histogram - visualising data
ggplot(mite, aes(x = Brachy)) +
  geom_histogram() +
  theme_classic()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Let's assume we want to check the abundance of Brachy (one of mite species) against substrate density and shrub levels. You can check the class of these with `str()`. We'll use a glm and the poisson family.

Modelling the data

```
mite_data <- cbind(mite["Brachy"], mite.env) # create a combined dataframe
# for ease of reference
head(mite_data)
```

```
##   Brachy SubsDens WatrCont Substrate Shrub    Topo
## 1     17    39.18    350.1   Sphagn1   Few Hummock
## 2      2    54.99    434.8    Litter   Few Hummock
## 3      4    46.07    371.7 Interface   Few Hummock
## 4     23    48.19    360.5   Sphagn1   Few Hummock
## 5      5    23.55    204.1   Sphagn1   Few Hummock
## 6     19    57.32    311.6   Sphagn1   Few Hummock
```

```
mod1 <- glm(data = mite_data,
            formula = Brachy ~ SubsDens * Shrub,
            family = poisson)
```

```
# run this for yourself to see the output:
summary(mod1)
```

If we look at our observations, we may wish to predict what will happen to the Brachy abundance if the plots with no shrubs (level = "none") increase in substrate density. Or any other number of possibilities.

```
#try runnning this yourself to see the plot
ggplot(mite_data, aes(x = Shrub, y = SubsDens))+
  geom_boxplot()+
  geom_point(position = position_jitter(.15))+
  theme_classic()
```

Setting up a new prediction dataframe

```
mite_new1 <- expand.grid(SubsDens = seq(from = min(mite_data$SubsDens),
                                        to = max(mite_data$SubsDens),
                                        by = 1),
                         Shrub = levels(mite_data$Shrub))

mite_new1$predicted_response_raw <- predict(object = mod1,
                                            newdata = mite_new1,
                                            type = "link")

mite_new1$predicted_response_transformed <- predict(object = mod1,
                                                    newdata = mite_new1,
                                                    type = "response")
```

Why are we using "response"? Glms are fitted using a link function. If you do not alter the defaults (we didn't, above), then the default family can be found at: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/family.html. You can extract the family used for your model using `family(mod1)`.

```
family(mod1) #link function
```

```
##
## Family: poisson
## Link function: log
```

```
mite_new1$check <- exp(mite_new1$predicted_response_raw) # this should end up the same
#as the "type = response"
```

Transforming to the response scale

So, we just backtransformed the type = "link" response in a new column "check", by applying the inverse link (for poisson, log link it is exp()). If our assumptions are correct, the `check` column should be the same as the `transformed_response` column we created earlier. [Recap: we exp(raw_response), or `type = "link"` column, and check this is the same as the `type = "response"`].

We can check whether is the same using `all.equal`

```
# ?all.equal # run this to see the results
all.equal(mite_new1$check, mite_new1$predicted_response_transformed)
```

```
## [1] TRUE
```

Creating confidence intervals using the inverse link

We will use this manual transformation method to calculate the confidence intervals on the "link" scale, and then back transform them to the response scale as we did above. The exact same method but we apply it to the confidence interval around the response, instead of the response.

Confidence intervals

We'll run through this as though you haven't already calculated the response (three times!).

```
#new dataframe:
mite_new1 <- expand.grid(SubsDens = seq(from = min(mite_data$SubsDens),
                                        to = max(mite_data$SubsDens),
                                        by = 1),
                         Shrub = levels(mite_data$Shrub))

#prediction - on the link scale
mite_new2 <- cbind(mite_new1,
                   predict(mod1, newdata = mite_new1, type = "link", se = TRUE))
head(mite_new2) #predict() as in the seminar has added the fit (untransformed response)
```

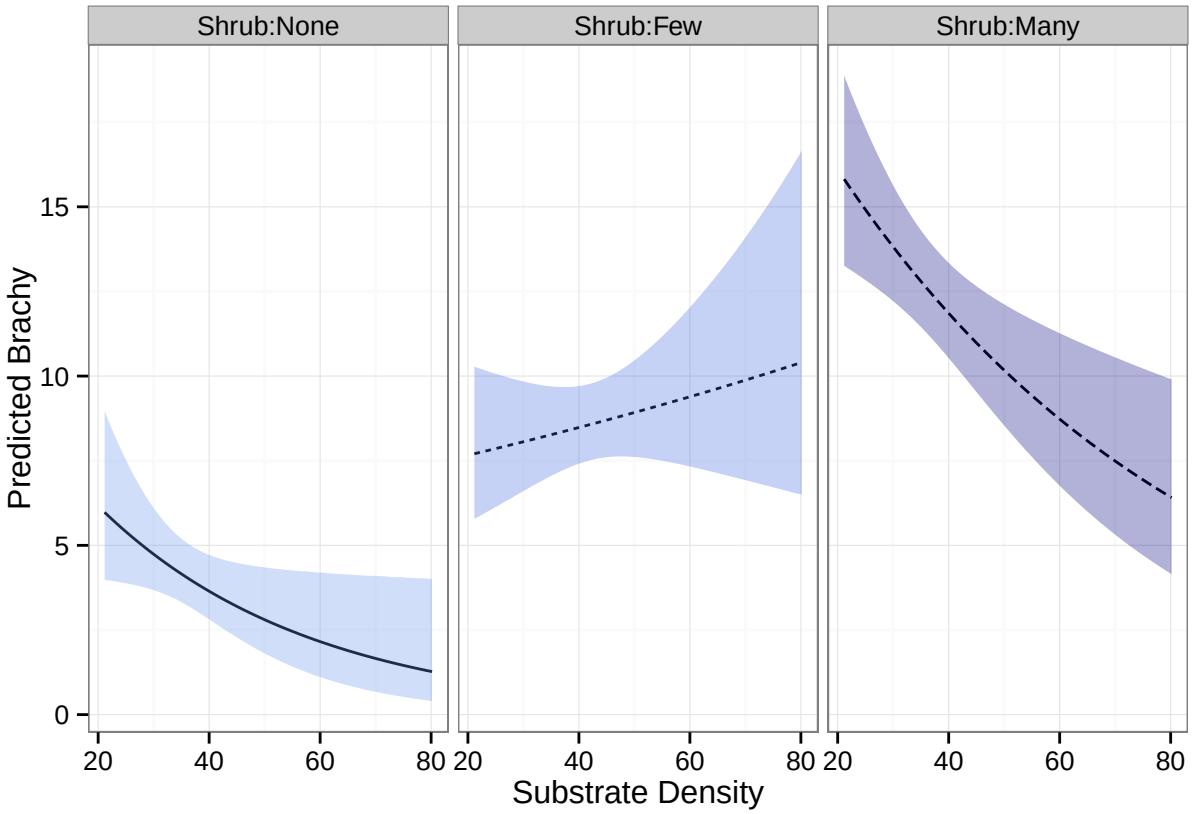
```
##   SubsDens Shrub   fit se.fit residual.scale
## 1    21.17  None 1.788 0.2073              1
## 2    22.17  None 1.761 0.1968              1
## 3    23.17  None 1.735 0.1866              1
## 4    24.17  None 1.709 0.1767              1
## 5    25.17  None 1.683 0.1672              1
## 6    26.17  None 1.656 0.1582              1
```

```
# the se.fit (untransformed se) and the residual scale (we are not using this).

#next, within the dataframe we just created, we do create new columns:
mite_new2 <- within(mite_new2, {
    transformed_response <- exp(fit) #back tranformaqtion as above
    lower_CI <- exp(fit - (1.96 * se.fit)) #backtransforming the SE
    upper_CI <- exp(fit + (1.96 * se.fit))
})


# to plot the results:
mite_new2$Shrub2 <- factor(mite_new2$Shrub,
                           levels = levels(mite_new2$Shrub),
                           labels = paste("Shrub", levels(mite_new2$Shrub), sep = ":"))

ggplot(mite_new2, aes(x = SubsDens, y = transformed_response))+
  geom_line(aes(linetype = Shrub))+
  geom_ribbon(aes(ymin = lower_CI, ymax = upper_CI, fill = Shrub), alpha = 0.3)+
  scale_fill_manual(values = c("cornflowerblue", "royalblue", "navy"))+
  labs(y = "Predicted Brachy", x = "Substrate Density")+
  guides(linetype = FALSE, fill = FALSE) +
  facet_wrap(~Shrub2) +
  theme_bw()
```



You'll see we use the "geom_ribbon" to make the transparent error bars, and use the pre-calculated ymin and ymax in it. You could also do `ymin = exp(fit - (1.96 * se.fit)), ymax = exp(fit + (1.96 * se.fit))` in the ggplot() command - up to you.

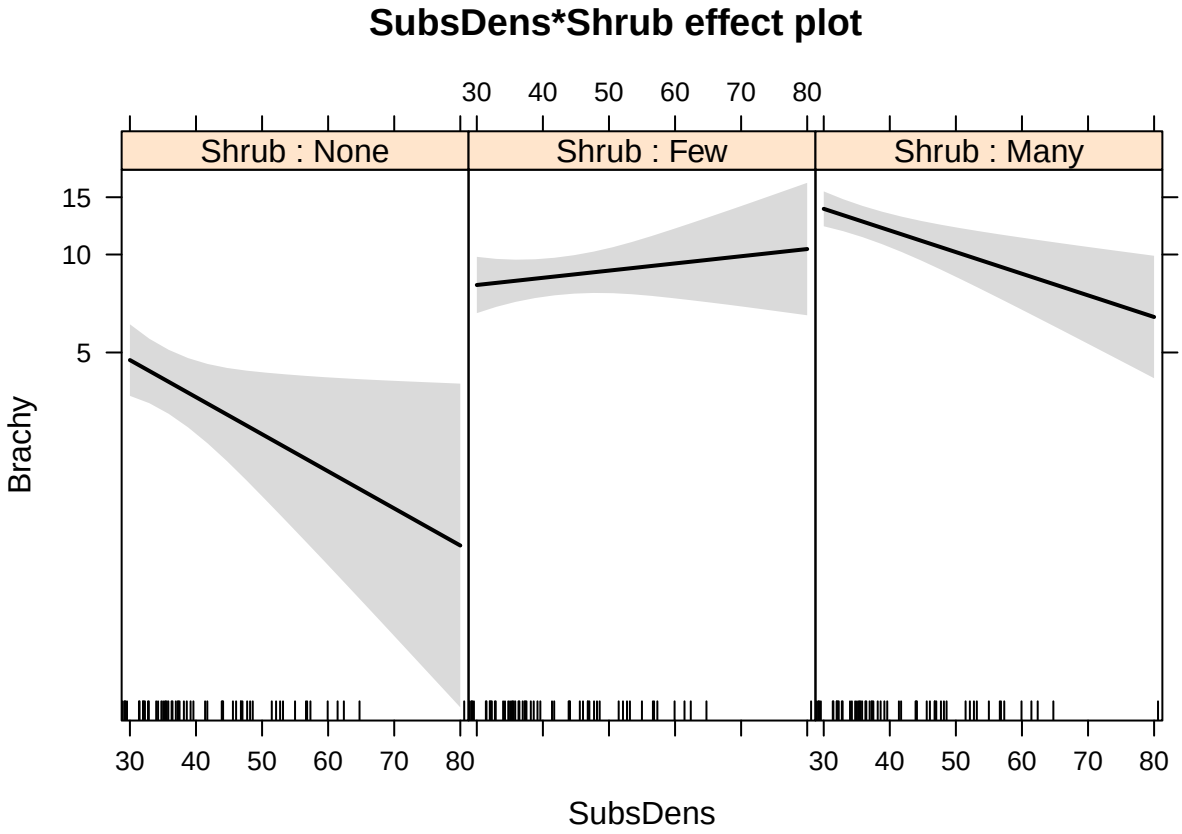
(2) Effects package

You'll need to install the effects package, if you haven't already, and then load it:

```
install.packages("effects")
library(effects)
```

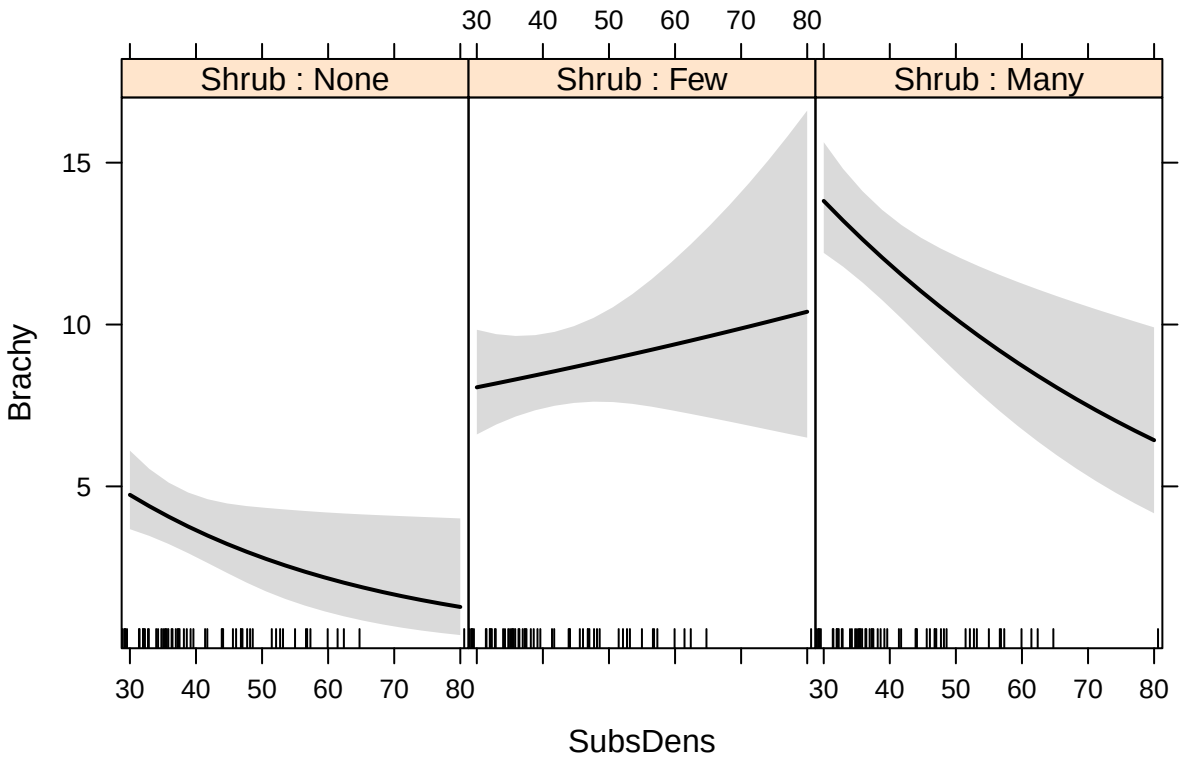
then it is simply a matter of doing the following, if plotting all the effects:

```
plot(allEffects(mod1))
```



Note this plots a similar standard error to that obtained by `se.fit=TRUE, type = "response"`, in that the confidence interval goes below zero which is unrealistic for count data. You need to use the `rescale = FALSE` option to avoid this.

```
plot(allEffects(mod1), rescale=F)
```


SubsDens*Shrub effect plot
Shrub : None
Shrub : Few
Shrub : Many
Brachy
SubsDens
30
40
50
60
70
80
5
10
15

(3) Coefplot2

To install coefplot2 you'll need to run the following, which may not work on uni computers (if they have banned you from downloading from certain websites):

```
install.packages("coefplot2", repos="http://www.math.mcmaster.ca/bolker/R", type="source")
```

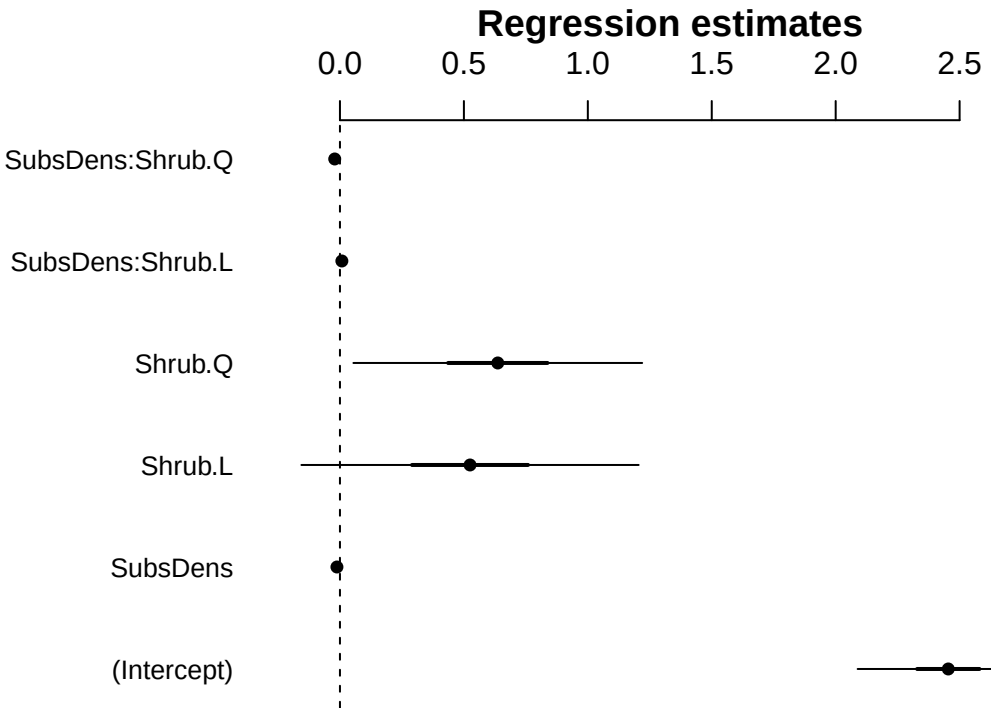
and then

```
library(coefplot2)
#or
require(coefplot2)

#these are, in general, the same thing.
```

We can use coefplot2 to visualise the *coefficients* in the model that are returned by `summary(mod1)`.

```
?coefplot2 #will run the help for coefplot2 and explain the concepts
coefplot2(mod1, intercept = TRUE)
```




```
# run summary(mod1) again and see how the coefficients
# and standard errors and CIs are transferred to the plot.
```

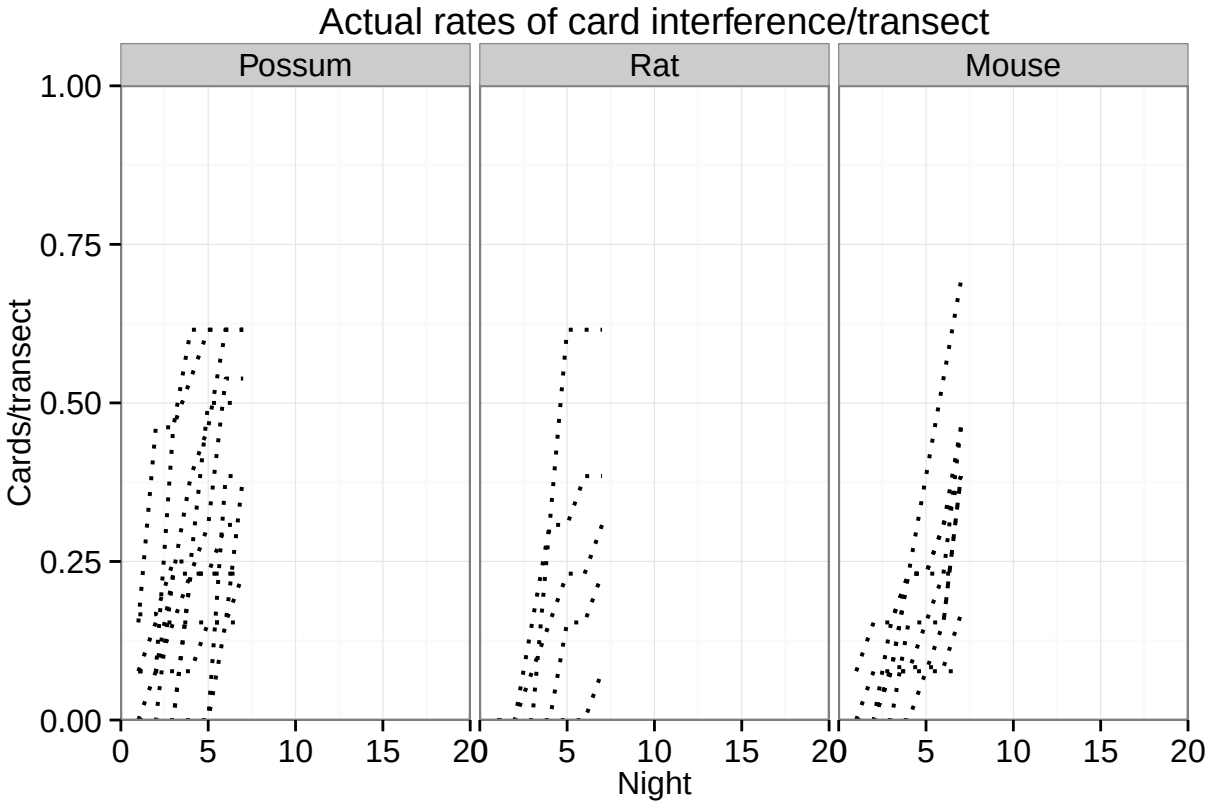
(4) Why do all the work then?

You may sometimes prefer finer control over the predictions, in which case you will probably find it easier to specify the new data yourself.

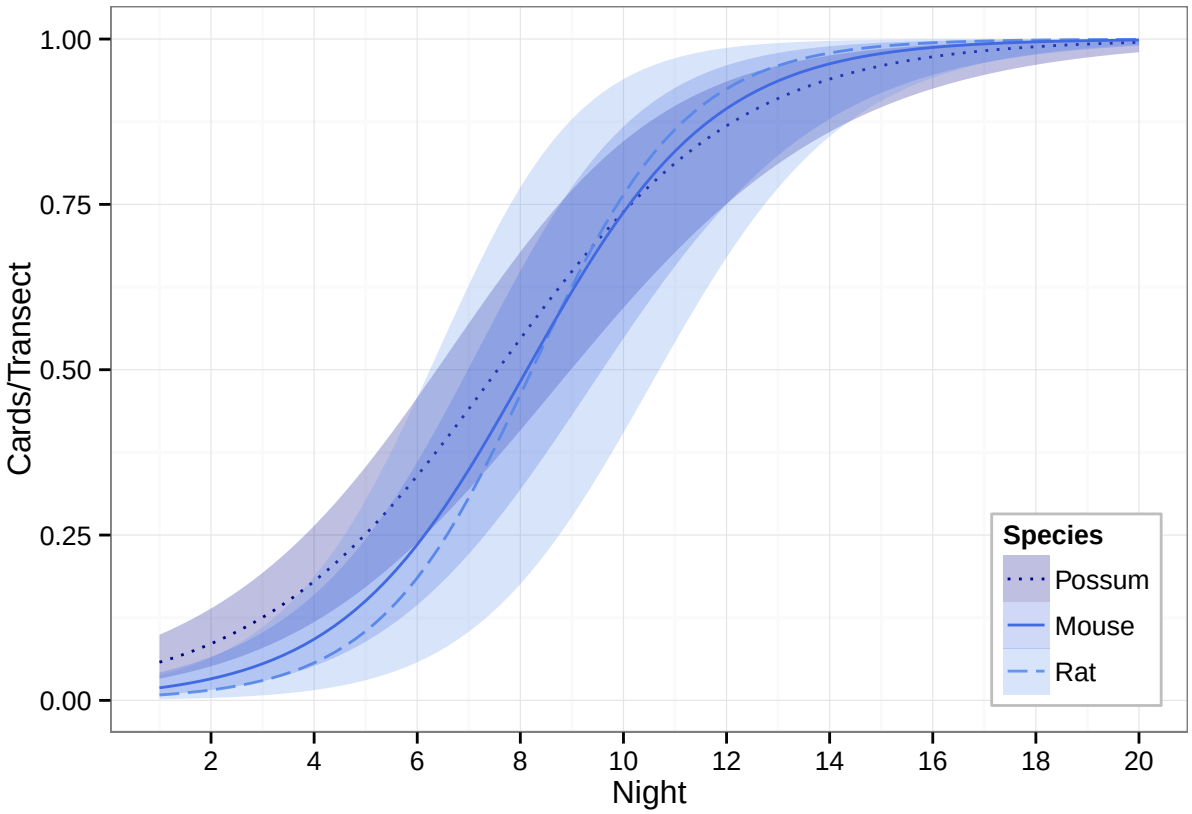
For example, for some recent analyses, I set out a number of monitoring stations ("CTCs") and checked how many of the total had been triggered each day. The family is therefore a binomial - the response (cumulative proportion of number of stations triggered).

I measured them for 7 nights, but I actually wanted to predict, based on the rates of trigger for the first 7 nights, what would have happened if they had been out for longer.

The raw data cumulative triggers looked like this:



But if I want to predict out 20 nights and have full control over the plot, I can manually predict as per the instructions above. The effects package is more for plotting effect sizes. The code is not shown here for brevity.



Go forth and predict!