Prediction and visualizing uncertainty: O-ring example

Brian Leung

2023-01-09

O-ring data

```
# load data
oring <- read_csv("https://www.openintro.org/data/csv/orings.csv")

# some wrangling
oring <-
  oring %>%
  mutate(damaged_dum = if_else(damaged >= 1, 1, 0)) %>%
  rename(temp = temperature)

head(oring)
```

```
## # A tibble: 6 x 5
##   mission  temp damaged undamaged damaged_dum
##     <dbl> <dbl>   <dbl>     <dbl>       <dbl>
## 1       1    53       5         1           1
## 2       2    57       1         5           1
## 3       3    58       1         5           1
## 4       4    63       1         5           1
## 5       5    66       0         6           0
## 6       6    67       0         6           0
```

A brief note on logistic regression

Consider the following logistic regression where we predict the probability of o-ring being damaged using temperature as the predictor:

$$\Pr(\text{Damage}|\text{Temp}) = \text{logit}^{-1}(\beta_0 + \beta_1 \text{Temp})$$

More generally, the *link function* that maps the linear predictor $X_i\beta$ to the probability π_i is logit in logistic regression, which is a *non-linear* transformation. We usually prefer to work with the inverse logit. The scale on which we're working is crucial in prediction:

$$\begin{aligned} \text{logit}(\pi_i) &= X_i\beta \\ \pi_i &= \text{logit}^{-1}(X_i\beta) \end{aligned}$$

Logit model on O-ring data

```r
# logit model
oring_logit <- glm(damaged_dum ~ temp, data = oring, family = "binomial")

# summary
summary(oring_logit)
```

```
##
## Call:
## glm(formula = damaged_dum ~ temp, family = "binomial", data = oring)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0611  -0.7613  -0.3783   0.4524   2.2175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  15.0429     7.3786   2.039   0.0415 *
## temp         -0.2322     0.1082  -2.145   0.0320 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 20.315  on 21  degrees of freedom
## AIC: 24.315
##
## Number of Fisher Scoring iterations: 5
```

```r
# regression table w/ stargazer
stargazer(oring_logit, type = "latex", header = FALSE)
```

Table 1:

	Dependent variable:
	damaged_dum
temp	-0.232^{**}
	(0.108)
Constant	15.043^{**}
	(7.379)
Observations	23
Log Likelihood	-10.158
Akaike Inf. Crit.	24.315
Note:	*p<0.1; **p<0.05; ***p<0.01

Why prediciton and visualization?

Logistical regression, despite its apparent simplicity and ubiquity, is notoriously hard to interpret directly:

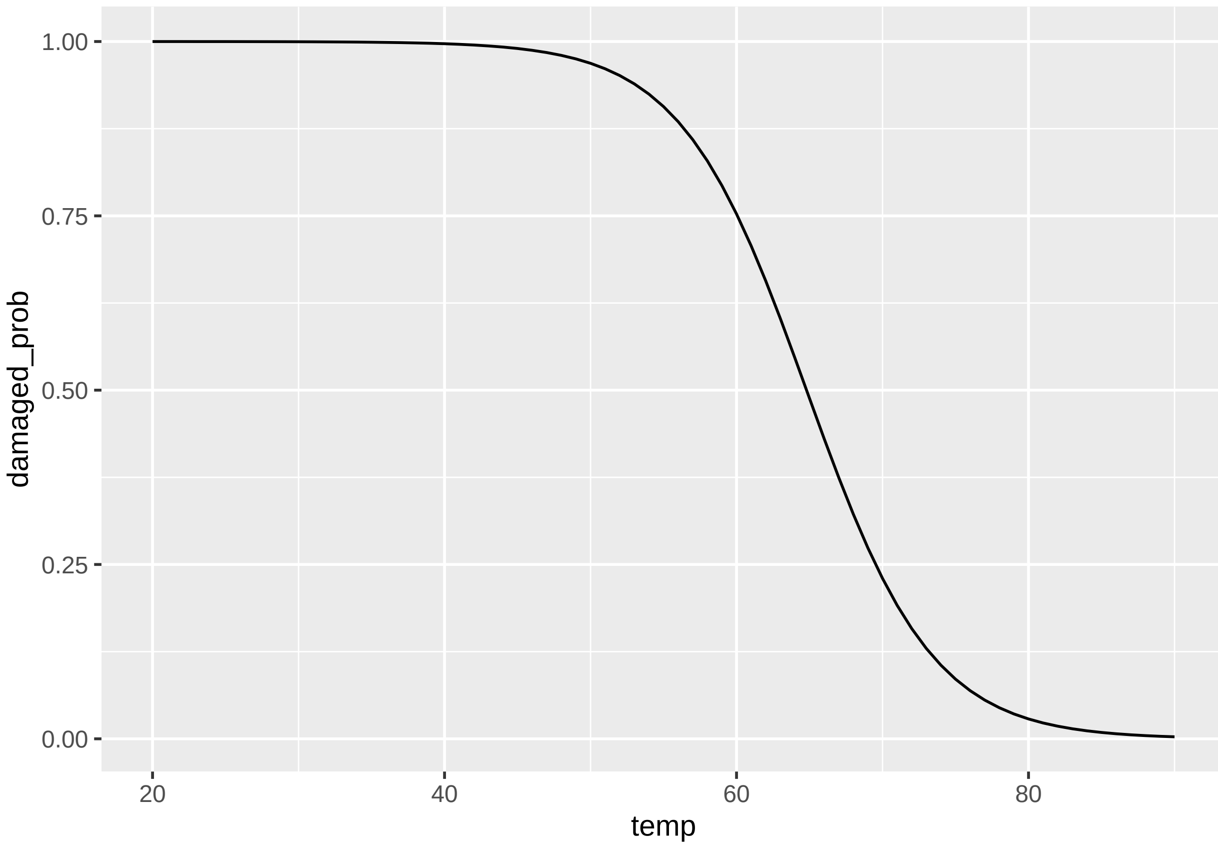
- Logit link function: For every unit increase in x_k, the log-odds ratio increases by β_k (???)
  - Exponentiation helps a bit, but not much…: For every unit increase in x_k, the odds ratio increases by e^{β_k}
- Non-linear nature of the link function: for models with multiple predictors, you can't directly interpret a single parameter
  - The slope on the logistic curve depends on your starting position
- Probabilities are much more interpretable and substantively meaningful
- But the problem of incorporating uncertainty into your prediction (e.g. computing confidence intervals)

Prediction w/ logit model

```
# create hypothetical values for temperature
temp_hypo <- tibble(temp = 20:90)

# predict prob of damage; be mindful of scale
damaged_prob <- predict(oring_logit, newdata = temp_hypo, type = "response")

# check the relationship b/w link and response
damaged_link <- predict(oring_logit, newdata = temp_hypo, type = "link")
all.equal(damaged_prob, inv.logit(damaged_link))
```

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

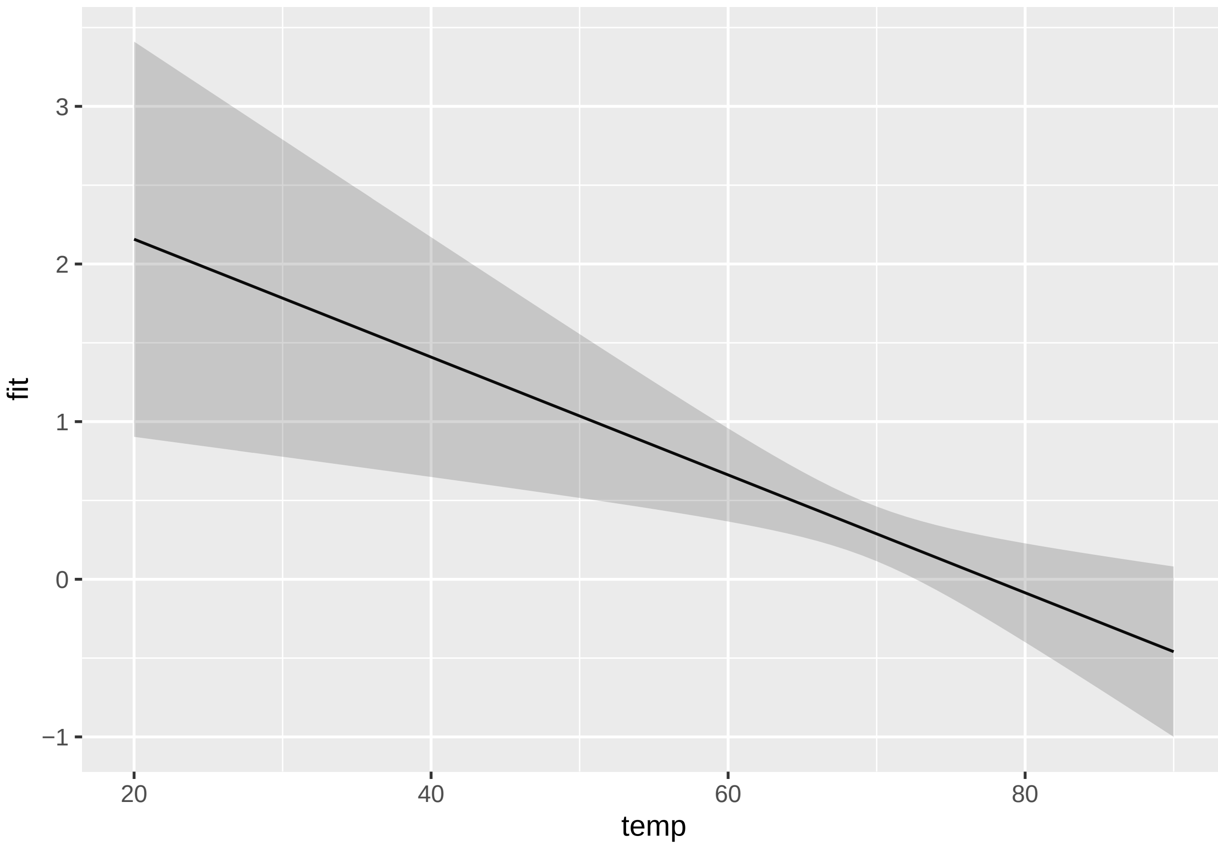
```
# merge prediction w/ hypo values
damaged_pred <- bind_cols(temp_hypo, damaged_prob = damaged_prob)

# visualize w/ ggplot2
ggplot(damaged_pred, aes(x = temp, y = damaged_prob)) +
  geom_line()
```



What is missing from the graph?

Confidence intervals: case of linear regression

```
# use linear regression instead
oring_lm <- lm(damaged_dum ~ temp, data = oring)
damaged_prob_lm <- predict(oring_lm, newdata = temp_hypo, interval = "confidence", level = 0.95)
damaged_pred_lm <- bind_cols(temp_hypo, damaged_prob_lm)

# visualize
ggplot(damaged_pred_lm, aes(x = temp, y = fit, ymin = lwr, ymax = upr)) +
  geom_line() +
  geom_ribbon(alpha = 0.2)
```



Confidence intervals: case of logit regression (or other GLMs)

```r
# predict doesn't work with glm objects in terms of calculating CIs
class(oring_logit)
```

```
## [1] "glm" "lm"
```

```r
predict(oring_logit, newdata = temp_hypo, interval = "confidence", level = 0.95)
```

```
##           1           2           3           4           5           6
## 10.39964676 10.16748402  9.93532127  9.70315853  9.47099579  9.23883304
##           7           8           9          10          11          12
##  9.00667030  8.77450755  8.54234481  8.31018207  8.07801932  7.84585658
##          13          14          15          16          17          18
##  7.61369383  7.38153109  7.14936834  6.91720560  6.68504286  6.45288011
##          19          20          21          22          23          24
##  6.22071737  5.98855462  5.75639188  5.52422913  5.29206639  5.05990365
##          25          26          27          28          29          30
##  4.82774090  4.59557816  4.36341541  4.13125267  3.89908993  3.66692718
##          31          32          33          34          35          36
##  3.43476444  3.20260169  2.97043895  2.73827620  2.50611346  2.27395072
##          37          38          39          40          41          42
##  2.04178797  1.80962523  1.57746248  1.34529974  1.11313699  0.88097425
##          43          44          45          46          47          48
##  0.64881151  0.41664876  0.18448602 -0.04767673 -0.27983947 -0.51200221
##          49          50          51          52          53          54
## -0.74416496 -0.97632770 -1.20849045 -1.44065319 -1.67281594 -1.90497868
##          55          56          57          58          59          60
## -2.13714142 -2.36930417 -2.60146691 -2.83362966 -3.06579240 -3.29795515
##          61          62          63          64          65          66
## -3.53011789 -3.76228063 -3.99444338 -4.22660612 -4.45876887 -4.69093161
##          67          68          69          70          71
## -4.92309436 -5.15525710 -5.38741984 -5.61958259 -5.85174533
```

Computing CIs for logit: inverse link function

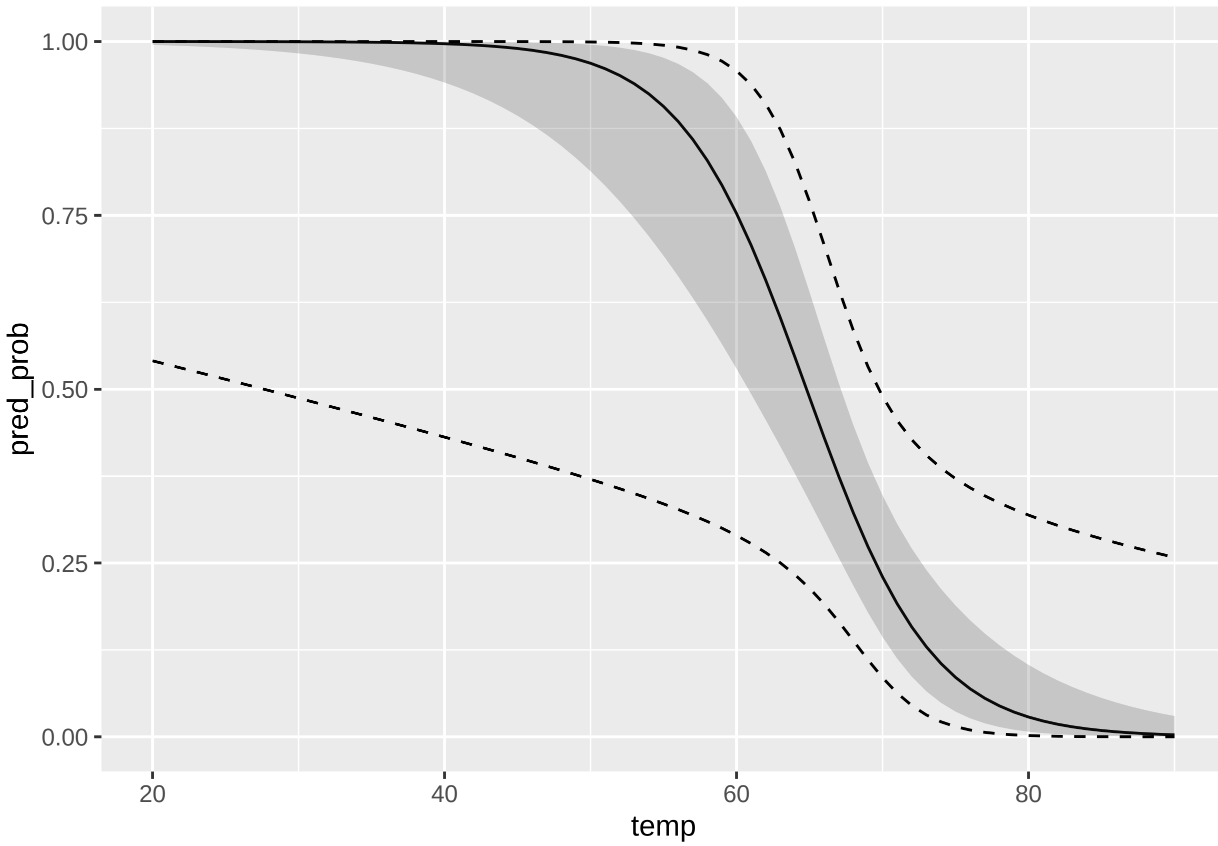
```r
# prediction on logit scale w/ standard errors
link_pred <- predict(oring_logit, newdata = temp_hypo, type = "link", se = TRUE)

# some wrangling
link_pred <-
  link_pred %>%
  bind_rows() %>%
  select(-residual.scale)

# critical values for 95% and 67% CIs; ignore problem of small-n for simplicity
qnorm(p = (1 - 0.95)/2, lower.tail = FALSE) # ~1.96
```

```
## [1] 1.959964
```

```r
qnorm(p = (1 - 0.67)/2, lower.tail = FALSE) # ~0.97
```

```
## [1] 0.9741139
```

```r
# manually compute CIs: transform linear predictor back to probability via inverse logit
link_pred_CIs <-
  link_pred %>%
  mutate(
    pred_prob = inv.logit(fit),
    upr_95 = inv.logit(fit + 1.96 * se.fit),
    lwr_95 = inv.logit(fit - 1.96 * se.fit),
    upr_67 = inv.logit(fit + 0.97 * se.fit),
    lwr_67 = inv.logit(fit - 0.97 * se.fit),
  ) %>%
  bind_cols(temp_hypo)

# visualize w/ ggplot2
link_pred_vis <-
 ggplot(link_pred_CIs, aes(x = temp, y = pred_prob, ymin = lwr_67, ymax = upr_67)) +
  geom_line() +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = upr_95), linetype = 2) +
  geom_line(aes(y = lwr_95), linetype = 2)

print(link_pred_vis)
```



1.00
0.75
0.50
0.25
0.00
pred_prob
20
40
60
80
temp

Computing CIs for logit: simulation method via MASS::mvrnorm()

Consult Chris's lecture on Maximum Likelihood for better reference.

Let's take a step back and think about `predict()` function: how does it calculate the predicted probability?

You can do it by hand. Quick example:

Let's say we want to know the probability of damage given that temperature is 50 degree. We know that the intercept coefficient is 15.0429 and the temperature coefficient is -0.2322.

$$
\begin{aligned}
\pi_{t=50} &= \text{logit}^{-1}(15.0429 \times 1 + -0.2322 \times 50) \\
&\approx 0.9687
\end{aligned}
$$

We can check the result with `predict()`

```r
predict(oring_logit, newdata = data.frame(temp = 50), type = "response")
```

```
##         1
## 0.9687735
```

But the problem is that we treat the estimated coefficients as certain and fail to *propagate uncertainty* from our estimation

How can we propagate uncertainty to our prediction? Counterfactual simulation!

Basic logic of counterfactual simulation:

1. Choose a set of counterfactual value for x_c
2. Estimate the model and obtain the parameter vector, $\hat{\beta}$, and its variance covariance matrix, $\hat{V}(\hat{\beta})$
3. Draw $\tilde{\beta}$ from the multivariate normal $f_{MVN}(\hat{\beta}, \hat{V}(\hat{\beta}))$
4. Calculate $\tilde{\pi}_c = \text{logit}^{-1}(x_c\tilde{\beta})$
5. Repeat the procedure many times, summarizing this vector to get expected values and confidence intervals

```r
# point estimate of the parameters
pe <- coef(oring_logit)

# variance covariance of the parameters
vc <- vcov(oring_logit)

# set N of simulations
sims <- 10000

# simulate many betas
sim_beta <- MASS::mvrnorm(sims, pe, vc)

dim(sim_beta)
```

```
## [1] 10000     2
```

Each row represents one trial in the simulation; there are 10,000 simulations, hence 10,000 rows.

Each column represents one simulated $\tilde{\beta}$; there are two parameters, hence 2 columns.

They encapsulate the uncertainties in our estimation.

Now we can calculate $\tilde{p_c}$ with matrix multiplication. To see this:

$$\underbrace{\begin{bmatrix} \tilde{\pi_{1,1}} & \tilde{\pi_{1,2}} & \dots & \tilde{\pi_{1,n}} \\ \tilde{\pi_{2,1}} & \tilde{\pi_{2,2}} & \dots & \tilde{\pi_{2,n}} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{\pi_{k,1}} & \tilde{\pi_{2,2}} & \dots & \tilde{\pi_{k,n}} \end{bmatrix}}_{\text{k counterfactuals; n simulations}} = \underbrace{\begin{bmatrix} 1 & x_{c,1} \\ 1 & x_{c,2} \\ \vdots & \vdots \\ 1 & x_{c,k} \end{bmatrix}}_{\text{k counterfactural; 2 variables}} \times \underbrace{\begin{bmatrix} \tilde{\beta_{0,1}} & \tilde{\beta_{0,2}} & \dots & \tilde{\beta_{0,n}} \\ \tilde{\beta_{1,1}} & \tilde{\beta_{1,2}} & \dots & \tilde{\beta_{1,n}} \end{bmatrix}}_{\text{2 parameters; n simulations}}$$

Intuitively, let's imagine there are $n = 10,000$ parallel universes, each of which is one simulation where $\tilde{\beta}$ exhibits some particular value (from a random MVN draw).

In each parallel universe (simulation), you calculate the particular $\tilde{\pi}$ for each and every counterfactual temparature $= \{20, 21, 22, \dots, 90\}$. Essentially, you're repeating the manual calculation we've done above for $k = 71$ times.

Then, you repeat the procedure for each and every parallel universe (simulation).

You should get $n \times k = 10,000 \times 71$ different $\tilde{\pi}$

```
# create hypothetical values for temp; plus constant
hypo_temp <- cbind(1, 20:90)

# check dimensions: 71
dim(hypo_temp)
```

```
## [1] 71  2
```

```
# check dimensions for simulated betas
dim(sim_beta)
```

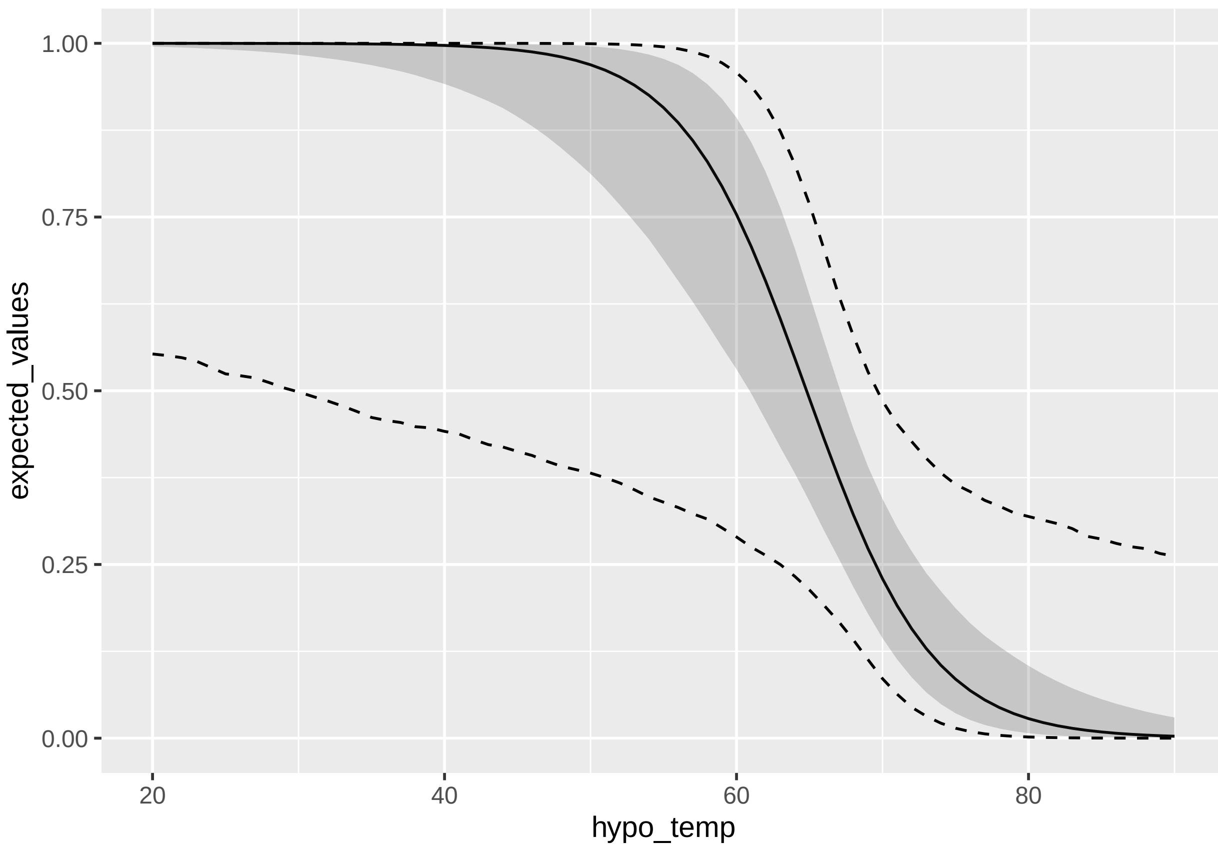
```
## [1] 10000     2
```

```
sim_beta <- t(sim_beta)

# matrix multiplication
sim_prob <- hypo_temp %*% sim_beta

# check dimensions for simulated probabilities
dim(sim_prob)
```

```
## [1]    71 10000
```

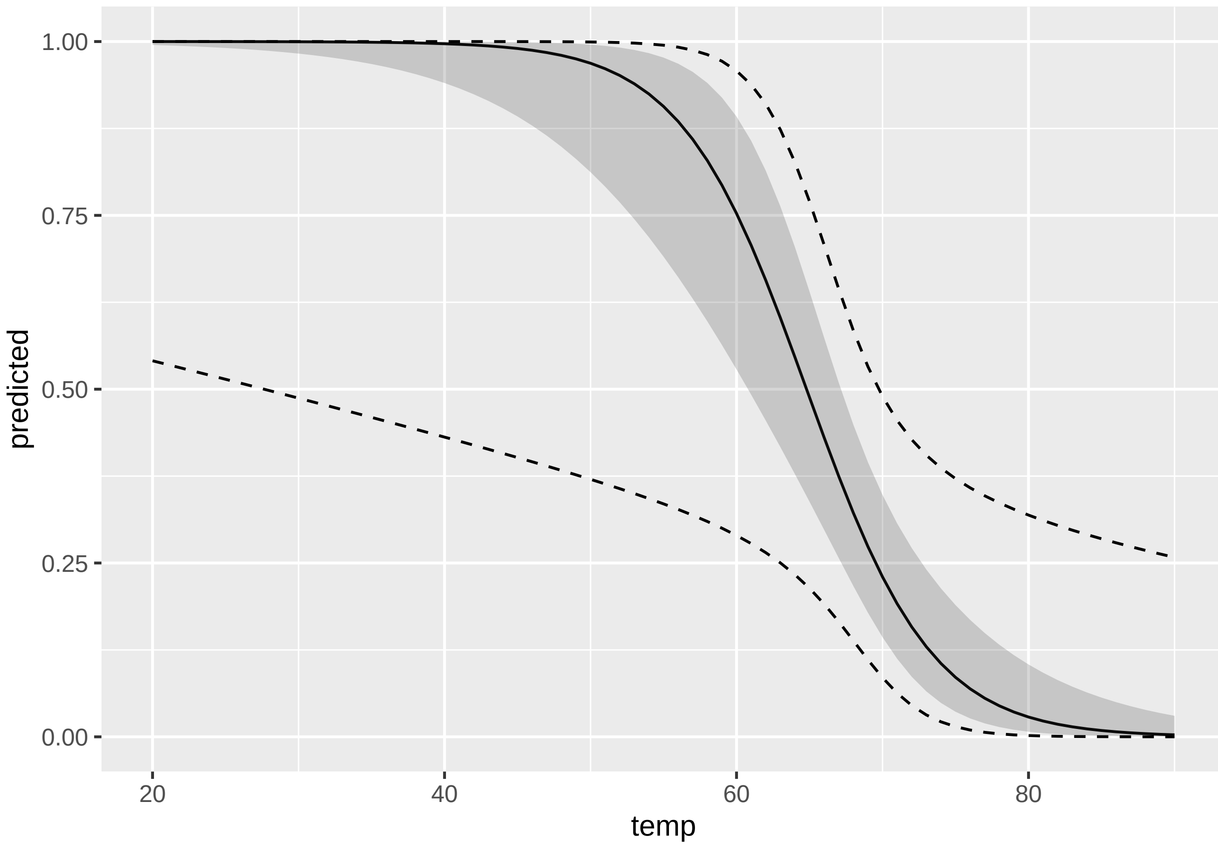
```
# calculate expected values via mean()
expected_values <- apply(sim_prob, 1, mean)

# calculate confidence intervals via quantile()
CIs_95 <- apply(sim_prob, 1, quantile, prob = c(0.025, 0.975))
CIs_67 <- apply(sim_prob, 1, quantile, prob = c(0.165, 0.835))

# put everything together
mvrnorm_sim_CIs <-
  bind_cols(expected_values = expected_values,
            t(CIs_95),
            t(CIs_67)) %>%
  mutate_all(inv.logit) %>%
  mutate(hypo_temp = 20:90)

# visualize w/ ggplot2
mvrnorm_sim_vis <-
```

```
  ggplot(mvrnorm_sim_CIs, aes(x = hypo_temp, y = expected_values, ymin = `16.5%`, ymax = `83.5%`)) +
  geom_line() +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = `97.5%`), linetype = 2) +
  geom_line(aes(y = `2.5%`), linetype = 2)

print(mvrnorm_sim_vis)
```



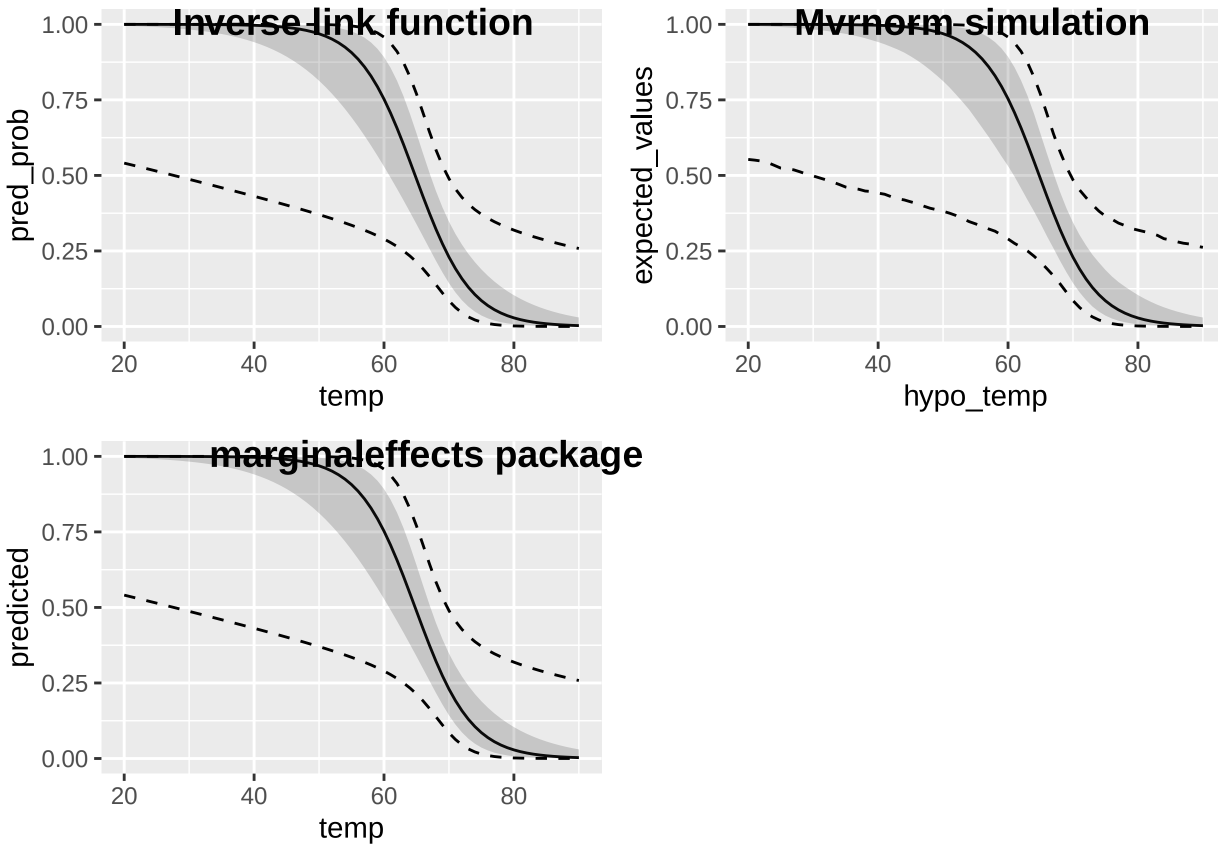
Computing CIs for logit: marginaleffects package

```
# use predictions() function from marginaleffects package
margin_pred_95 <- predictions(oring_logit, newdata = datagrid(temp = 20:90), conf_level = 0.95)
margin_pred_67 <- predictions(oring_logit, newdata = datagrid(temp = 20:90), conf_level = 0.67)

# some wrangling
margin_pred_95 <-
  margin_pred_95 %>%
  select(temp, predicted, conf.low, conf.high) %>%
  rename(conf_low_95 = conf.low,
         conf_high_95 = conf.high)

margin_pred_67 <-
  margin_pred_67 %>%
  select(conf.low, conf.high) %>%
  rename(conf_low_67 = conf.low,
         conf_high_67 = conf.high)

# put everything together
margin_pred_CIs <- cbind(margin_pred_95, margin_pred_67)

# visualize w/ ggplot2
margin_pred_vis <-
  ggplot(margin_pred_CIs, aes(x = temp, y = predicted, ymin = conf_low_67, ymax = conf_high_67)) +
  geom_line() +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = conf_low_95), linetype = 2) +
  geom_line(aes(y = conf_high_95), linetype = 2)

print(margin_pred_vis)
```


1.00
0.75
0.50
0.25
0.00
predicted
20
40
60
80
temp

Computing CIs for logit: compare all three methods

```r
# use plot_grid() function from cowplot
plot_grid(link_pred_vis, mvrnorm_sim_vis, margin_pred_vis,
          labels = c("Inverse link function", "Mvrnorm simulation", "marginaleffects package"))
```



Final remarks

- It's reassuring that all three methods produce equivalent results
- Hazards of over-relying on off-the-shelf functions or packages: opaque computation can produce unintended, or often wrong, results
  - Especially when your models become more complex and with more variables
- Manual simulations can be flexible: e.g. computing first difference and its uncertainties
  - Given a 10 degree increase in temperature, what is the change in probabilities in damage (and its uncertainties)
  - Also, a great conceptual check on your fundamental understanding of regression
- We didn't talk about how to improve the graphs visually
  - Ugly defaults; no annotation
  - Also, there are more to the inner working of `ggplot2`
  - After the lectures have covered more on scientific principles on visual displays, we'll return to this example

Knitting PDF

You have to install `tinytex` before you can knit a PDF file. Run the following code. We'll talk about LaTeX next week.

```r
# install.packages("tinytex")
# tinytex::install_tinytex()
```