Prediction and visualizing uncertainty: O-ring example

Brian Leung

2023-01-09

O-ring data

```r
# 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:

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

More generally, the link function that maps the linear predictor \(X_i \beta\) to the probability \(\pi_i\) 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:

\[
\logit(\pi_i) = X_i \beta
\]

\[
\pi_i = \logit^{-1}(X_i \beta)
\]
Logit model on O-ring data

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

# summary
summary(oring_logit)
```

### Call:
```
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 *
```

### (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

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dependent variable:</strong></td>
<td></td>
</tr>
<tr>
<td>damaged_dum</td>
<td></td>
</tr>
<tr>
<td>temp</td>
<td>-0.232**</td>
</tr>
<tr>
<td></td>
<td>(0.108)</td>
</tr>
<tr>
<td>Constant</td>
<td>15.043**</td>
</tr>
<tr>
<td></td>
<td>(7.379)</td>
</tr>
<tr>
<td>Observations</td>
<td>23</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-10.158</td>
</tr>
<tr>
<td>Akaike Inf. Crit.</td>
<td>24.315</td>
</tr>
</tbody>
</table>

*Note:* *p<0.1; **p<0.05; ***p<0.01
Why prediction 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 \( \beta_k \) (???)
  
  - Exponentiation helps a bit, but not much...: For every unit increase in \( x_k \), the odds ratio increases by \( e^{\beta_k} \)

- 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

```r
# 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

```r
# 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"

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>6.22071737</td>
<td>5.98855462</td>
<td>5.75639188</td>
<td>5.52422913</td>
<td>5.29206639</td>
<td>5.05990365</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>3.43476444</td>
<td>3.20260169</td>
<td>2.97043895</td>
<td>2.73827620</td>
<td>2.50611346</td>
<td>2.27395072</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>2.04178797</td>
<td>1.80962523</td>
<td>1.57746248</td>
<td>1.34529974</td>
<td>1.11313699</td>
<td>0.88097425</td>
</tr>
<tr>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>0.64881151</td>
<td>0.41664876</td>
<td>0.18448602</td>
<td>-0.04767673</td>
<td>-0.27983947</td>
<td>-0.51200221</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>-0.74416496</td>
<td>-0.97632770</td>
<td>-1.20849045</td>
<td>-1.44065319</td>
<td>-1.67281594</td>
<td>-1.90497868</td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>-4.92309436</td>
<td>-5.15525710</td>
<td>-5.38741984</td>
<td>-5.61958259</td>
<td>-5.85174533</td>
<td></td>
</tr>
</tbody>
</table>
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
qnorm(p = (1 - 0.67)/2, lower.tail = FALSE) # ~0.97
## [1] 0.9741139

# 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)
```
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.

\[
\pi_{t=50} = \logit^{-1}(15.0429 \times 1 + -0.2322 \times 50)
\]
\[
\approx 0.9687
\]

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 \(\tilde{\pi}_c = \logit^{-1}(x_c \tilde{\beta})\)
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 \(\tilde{\pi}_c\) with matrix multiplication. To see this:
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 temperature \( = \{20, 21, 22, \ldots, 90\} \). 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 \( \tilde{\pi} \)

```r
# 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

```r
# 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)
```
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()
```