Prediction and visualizing uncertainty: O-ring example

Brian Leung

2024-01-05

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:

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

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:
## glm(formula = damaged_dum ~ temp, family = "binomial", data = oring)

## 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
# type = "text" for in-console display
# type = "latex" for knitting PDF
stargazer(oring_logit, type = "latex", header = FALSE)

<table>
<thead>
<tr>
<th>Dependent variable:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>damaged_dum</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>-0.232** (0.108)</td>
</tr>
<tr>
<td>Constant</td>
<td>15.043** (7.379)</td>
</tr>
</tbody>
</table>

| Observations | 23 |
| Log Likelihood | -10.158 |
| Akaike Inf. Crit. | 24.315 |

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:** How to interpret the coefficients?
  - 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 initial position
- **Probabilities are much more interpretable and substantively meaningful**
- **Plus 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")

inv.logit <- plogis
all.equal(damaged_prob, inv.logit(damaged_link))
```

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

```r
# 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)

# predict doesn't work with glm objects in terms of calculating CIs

```r
class(oring_logit)
```

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

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

```
#   1     2     3     4     5     6
#   7     8     9    10    11    12
#   13    14    15    16    17    18
#   19    20    21    22    23    24
## 6.22071737 5.98855462 5.75639188 5.52422913 5.29206639 5.05990365
#   25    26    27    28    29    30
#   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 -0.2793947 -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
#   61    62    63    64    65    66
#   67    68    69    70    71    72
## -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
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()


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")
```

```r
## 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 \(f_{MVN}(\tilde{\beta}, \hat{V}(\tilde{\beta}))\)
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

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

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

# set N of simulations
sims <- 1000

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

dim(sim_beta)
```

```r
## [1] 1000  2
```

Each row represents one trial in the simulation; there are 1,000 simulations, hence 1,000 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 \( n = 1,000 \) 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 \( n \times k = 1,000 \times 71 \) different \( \tilde{\pi} \)

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

# check dimensions: 71 rows, 2 columns
dim(hypo_temp)
## [1] 71  2

# check dimensions for simulated betas
dim(sim_beta)
## [1] 1000  2

sim_beta <- t(sim_beta)

# matrix multiplication
sim_prob <- hypo_temp %*% sim_beta

# check dimensions for simulated probabilities
dim(sim_prob)
## [1] 71 1000

# 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_95 <- t(CIs_95)

CIs_67 <- apply(sim_prob, 1, quantile, prob = c(0.165, 0.835))
CIs_67 <- t(CIs_67)

# put everything together
mvnorm_sim_CIs <-
  bind_cols(expected_values = expected_values,
            CIs_95, 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 %>%
  as_tibble() %>%
  select(temp, estimate, conf.low, conf.high) %>%
  rename(conf_low_95 = conf.low,
         conf_high_95 = conf.high)

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

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

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

print(margin_pred_vis)
```
Computing CIs for logit: compare all three methods

# 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"))

---

**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()
```