Untitled

Nikhil Gopal

2/7/2022

```
rm(list = ls())
library(arm)
library(rstanarm)
```

13.4

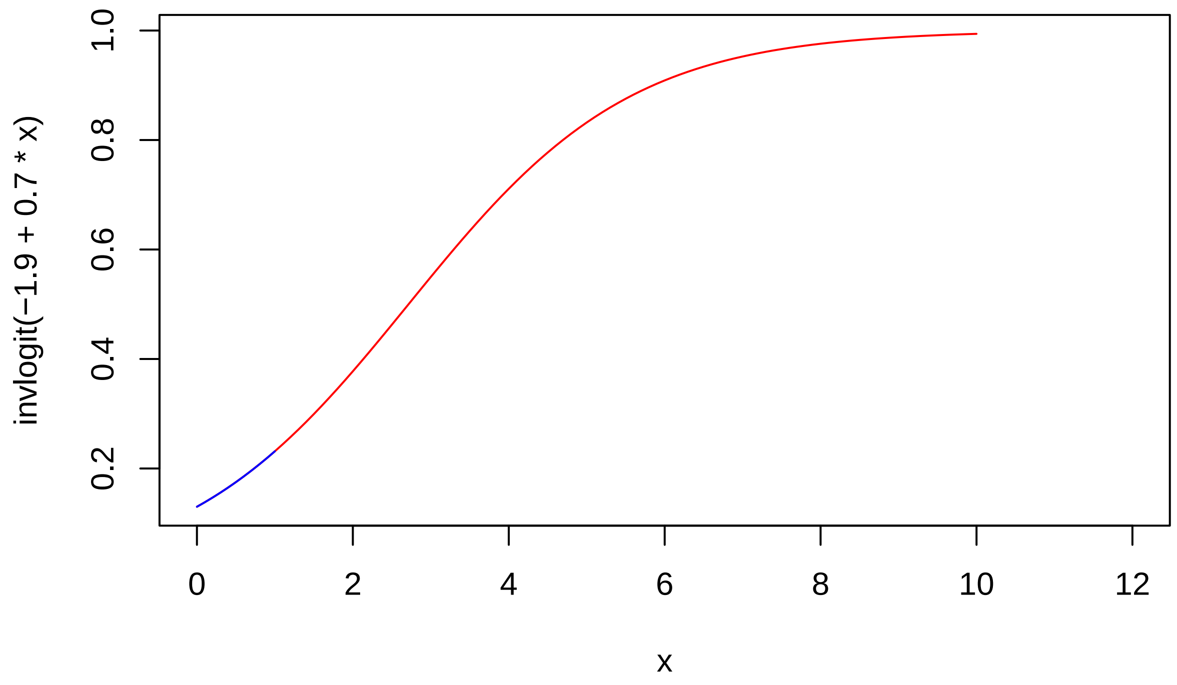
Logistic regression with two predictors: The following logistic regression has been fit:

Median MAD_SD (Intercept) -1.9 0.6 x 0.7 0.8 z 0.7 0.5

Here, x is a continuous predictor ranging from 0 to 10, and z is a binary predictor taking on the values 0 and 1. Display the fitted model as two curves on a graph of Pr(y = 1) vs. x

In the graph below, the blue line represents the z predictor as it is a binary variable that only goes from 0-1, and the red line which goes from 0 10 represents the x predictor.

```
curve(invlogit(-1.9 + 0.7 * x), xlim = c(0, 12), from = 0, to = 10, col = "red")
curve(invlogit(-1.9 + 0.7 * x), xlim = c(0, 12), from = 0, to = 1, add = TRUE, col = "blue")
```



```
#add statement allows both to be shown on the same graph
```

13.5a

Interpreting logistic regression coefficients: Here is a fitted model from the Bangladesh analysis predicting whether a person with high-arsenic drinking water will switch wells, given the arsenic level in their existing well and the distance to the nearest safe well:

stan_glm(formula = switch ~ dist100 + arsenic, family=binomial(link="logit"), data=wells)

Median MAD_SD (Intercept) 0.00 0.08 dist100 -0.90 0.10 arsenic 0.46 0.04

Compare two people who live the same distance from the nearest well but whose arsenic levels differ, with one person having an arsenic level of 0.5 and the other person having a level of 1.0. You will estimate how much more likely this second person is to switch wells. Give an approximate estimate, standard error, 50% interval, and 95% interval, using two different methods:

Use the divide-by-4 rule, based on the information from this regression output

The divide by 4 rule is a helpful approximation in logistic regression to quickly gain the upper bound on the limit of the probability of y = 1 given a 1 unit increase in predictor values by dividing logistic coefficient values by 4. Since we are interested in a unit change of 0.5, we will multiply these values by 0.5. A 50% confidence interval can be found by multiplying coefficient value and adding/subtracting 0.67449*1 standard error, while 95% can be found by adding/subtracting 2 standard errors.

```
distance <- -0.90/4
distance <- distance * 0.5
distance_sterr <- 0.10/4 * 0.5
```

```r
arsenic <- 0.46/4
arsenic <- arsenic * 0.5
arsenic_sterr <- 0.04/4 * 0.5
```

This gives us a final estimate for the probability of y = 1 given a 0.5 unit increase for arsenic to be 0.0575. The standard error is 0.005. Thus, a 50% confidence interval is [0.0541, 0.0609], while the 95% confidence interval is [0.0475, 0.0675].

13.5b

Use predictive simulation from the fitted model in R, under the assumption that these two people each live 50 meters from the nearest safe well.

```r
#fit model
bangladesh_wells <- stan_glm(formula = switch ~ dist + arsenic, family=binomial(link="logit"),
          data=wells)
```

```
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.7 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.335619 seconds (Warm-up)
## Chain 1:                0.33556 seconds (Sampling)
## Chain 1:                0.671179 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.56 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.328395 seconds (Warm-up)
## Chain 2:                0.372346 seconds (Sampling)
## Chain 2:                0.700741 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.57 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.336088 seconds (Warm-up)
## Chain 3:                0.363665 seconds (Sampling)
## Chain 3:                0.699753 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 5.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.52 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.337081 seconds (Warm-up)
## Chain 4:                0.345278 seconds (Sampling)
## Chain 4:                0.682359 seconds (Total)
## Chain 4:
```

```
#create new dfs to represent cases we want to predict

ars_0.5 <- new <- data.frame(dist = 0.5, arsenic = 0.5)
ars_1 <- new <- data.frame(dist = 0.5, arsenic = 1)



#make predictions
ars_0.5$predicted <- predict(bangladesh_wells, type="response", newdata=ars_0.5, se.fit = T)$fit
ars_0.5$predicted_se <- predict(bangladesh_wells, type="response", newdata=ars_0.5, se.fit = T)$se.fit
ars_1$predicted <- predict(bangladesh_wells, type="response", newdata=ars_1, se.fit = T)$fit
ars_1$predicted_se <- predict(bangladesh_wells, type="response", newdata=ars_1, se.fit = T)$se.fit
difference <- ars_1$predicted - ars_0.5$predicted
difference_se <- ars_1$predicted_se - ars_0.5$predicted_se

print(paste("Arsenic = 0.5 y predicted value: ",round(ars_0.5$predicted, 4), "SE: ", round(ars_0.5$predi
```

```
## [1] "Arsenic = 0.5 y predicted value:  0.5571 SE:  0.0168"
```

```
print(paste("Arsenic = 1 y predicted value: ",round(ars_1$predicted, 4), "SE: ", round(ars_1$predicted_
```

```
## [1] "Arsenic = 1 y predicted value:  0.6129 SE:  0.0148"
```

```
print(paste("Difference: ",  round(difference, 4)))
```

```
## [1] "Difference:  0.0558"
```

```
print(paste("Difference (SE): ",  round(difference_se, 4)))
```

```
## [1] "Difference (SE):  -0.0021"
```

With a predictive simulation, the expected difference in the probability of switch, per unit change in arsenic, is 0.0574 with a standard error of 0.003. This gives a corresponding 50% confidence interval of [0.0554, 0.0594] and a 95% confidence interval of [0.0514, 0.0634].