Homework 9

Brady Lamson

2022-04-20

Chapter 13

Problem 6

Sally and Joan plan to meet to study in their college campus center. They are both impatient people who will only wait 10 minutes for the other before leaving. Rather than pick a specific time to meet, they agree to head over to the campus center sometime between 7:00 and 8:00 pm. Let both arrival times be normally distributed with mean 30 minutes past and a standard deviation of 10 minutes. Assume that they are independent of each other. What is the probability that they actually meet? Estimate the answer using simulation techniques introduced in this chapter, with at least 10,000 simulations.

```r
n <- 10000
sim_meet <- tibble::tibble(
    sally = rnorm(n, mean = 30, sd = 10),
    joan = rnorm(n, mean = 30, sd = 10),
    result = ifelse(
        abs(sally - joan) <= 10, "They meet", "They do not"
    )
)
mosaic::tally(~ result, format = "percent", data = sim_meet)
```

```
## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2
```

```
## result
## They do not   They meet
##       48.93       51.07
```

```r
mosaic::binom.test(~result, n, success = "They meet", data = sim_meet)
```

```
##
##
##
## data:  sim_meet$result  [with success = They meet]
## number of successes = 5107, number of trials = 10000, p-value = 0.03317
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.5008505 0.5205433
```

```
## sample estimates:
## probability of success
##                 0.5107
```

Problem 9

Generate $n = 5000$ observations from a logistic regression model with parameters intercept $\beta_0 = -1$, slope $\beta_1 = 0.5$, and distribution of the predictor being normal with mean $\mu = 1$ and standard deviation $\sigma = 1$. Calculate and interpret the resulting parameter estimates and confidence intervals.

```r
set.seed(500)

# Generate 5000 predictor variables
x <- rnorm(n = 5000, mean = 1, sd = 1)

# Set beta values
beta_0 <- -1
beta_1 <- 0.5

# Generate set of probabilities
true_probs <- exp(beta_0 + beta_1 * x) / (1 + exp(beta_0 + beta_1 * x))

# Feed probabilities into a binomial function
y <- rbinom(n = 5000, size = 1, prob = true_probs)

# Generate data frame using predictor and probability values
sim_data <- dplyr::tibble(X = x, Y = y)
```

```r
log_reg <- glm(Y ~ X, data = sim_data, family = "binomial")

log_summary <- summary(log_reg)
log_summary
```

```
##
## Call:
## glm(formula = Y ~ X, family = "binomial", data = sim_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6941  -0.9747  -0.7874   1.2542   2.0241
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.99669    0.04572  -21.80   <2e-16 ***
## X            0.49084    0.03173   15.47   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6644.5  on 4999  degrees of freedom
## Residual deviance: 6387.0  on 4998  degrees of freedom
## AIC: 6391
##
## Number of Fisher Scoring iterations: 4
```

```r
# log_summary$coefficients holds the data frame for the coefficients.
# Column 1 is the estimated values, column 2 is the standard errors

beta_0_est <- log_summary$coefficients[1,1]
beta_0_error <- log_summary$coefficients[1,2]
beta_1_est <- log_summary$coefficients[2,1]
beta_1_error <- log_summary$coefficients[1,2]
```

```r
# Create function to help calc intervals

calc_ci <- function(estimate, error) {
    low <- estimate - (2 * error)
    high <- estimate + (2 * error)

    c(low, high)
}
```

```r
# Calculate confidence intervals and round

beta_0_ci <- calc_ci(beta_0_est, beta_0_error) %>% round(3)
beta_1_ci <- calc_ci(beta_1_est, beta_1_error) %>% round(3)
```

```
## The confidence interval for beta 0 is [-1.088, -0.905].
## The confidence interval for beta 1 is [0.399, 0.582].
```

The parameter estimates were largely spot on! The confidence intervals are relatively small and neatly capture the true parameter value.

Chapter 19

Problem 3

Note: Copying and pasting these into this messes with the formatting of this page a bit. I'm not sure how to fix that!

Guesses:

- 1: str_subset(x, pattern = "pop")
  - This will return the first 7 words in the vector, specifically because they all have "pop" in them.
- 2: str_detect(x, pattern = "^pop")
  - This will return a vector of booleans, the first 6 being TRUE. The rest will be FALSE.
- 3: str_detect(x, pattern = "populari[sz]e")
  - Index 3 and 4 will be TRUE, the rest of the vector will be FALSE.
- 4: str_detect(x, pattern = "pop.*e")
  - This looks for words containing some combination of [pop…e]. Indices 3, 4, 7 will be TRUE, the rest FALSE.
- 5: str_detect(x, pattern = "p[a-z]*e")
  - This looks for any number of any letter contained between a "p" and an "e". Indices 3, 4, 7, 8 will be TRUE, the rest FALSE (though im unsure of index 10 here).
- 6: str_detect(x, pattern = "^[Pp][a-z]+.*n")
  - This looks for words starting with either "P" or "p", followed by any letter. The plus indicates any number of letters, followed by any number of any type of character. Finally, we end with an n. This applies to indices 6 and 12. So those will be TRUE, the rest FALSE.
- 7: str_subset(x, pattern = "^[^Pp]")
  - This just looks for words starting with anything BUT "P" or "p". So this returns indices 7, 8, 9 and 11.
- 8: str_detect(x, pattern = "^[A-Za-p]")
  - This looks for words starting with either a capital letter or a lowercase letter. That applies to everything BUT index 10. So index 10 is TRUE, all the rest are false.
- 9: str_detect(x, pattern = "[ ]")
  - This just looks for a space character which is found in indices 9, 10 and 11. Those three will be TRUE, the rest FALSE.
- 10: str_subset(x, pattern = "[]")
  - This looks for a tab. One is found on index 10, so that is the only string returned by this.
- 11: str_detect(x, pattern = "[ ]")

  - Same as last time, but this also wants a space before the tab. Nothing meets that criteria, so this will return a vector of only FALSE values.

- 12: str_subset(x, pattern = “[1]”)

  - This returns a vector containing any string starting with a space, which is index 11.

Running commands:

```
x <- c("popular", "popularity", "popularize", "popularise", "Popular",
       "population", "repopulate", "reproduce", "happy family",
       "happier\tfamily", " happy family", "P6dn")

str_subset(x, pattern = "pop")                    #1
```

```
## [1] "popular"    "popularity" "popularize" "popularise" "population"
## [6] "repopulate"
```

```
str_detect(x, pattern = "^pop")                   #2
```

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

```
str_detect(x, pattern = "populari[sz]e")          #3
```

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

```
str_detect(x, pattern = "pop.*e")                 #4
```

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

```
str_detect(x, pattern = "p[a-z]*e")               #5
```

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

```
str_detect(x, pattern = "^[Pp][a-z]+.*n")         #6
```

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

```
str_subset(x, pattern = "^[^Pp]")                 #7
```

```
## [1] "repopulate"      "reproduce"       "happy family"    "happier\tfamily"
## [5] " happy family"
```

```
str_detect(x, pattern = "^[A-Za-p]")              #8
```

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

[1]

```r
str_detect(x, pattern = "[ ]")                    #9
```

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

```r
str_subset(x, pattern = "[\t]")                   #10
```

```
## [1] "happier\tfamily"
```

```r
str_detect(x, pattern = "[ \t]")                  #11
```

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

```r
str_subset(x, pattern = "^[ ]")                   #12
```

```
## [1] " happy family"
```

Problem 4

Use the `babynames` data table from the `babynames` package to find the 10 most popular:

```
- a: Boys names ending in a vowel.

- b: Names ending with `joe`, `jo`, `Joe`, `Jo` (e.g., Billyjoe).
```

```r
# A
# Firstly I want to make sure I don't save babynames as a variable. It's HUGE, so we pipe
# it in directly. Secondly we filter using str_detect, putting all the vowels in brackets.
# Include sex requirement as well. From there you just group the names together and sum up
# the counts for all of the years.
# Sort in descending order and restrict to the top 10 and you're done!

babynames::babynames %>%
    filter(
        stringr::str_detect(string = name, pattern = "[AEIOUaeiou]$") & sex == "M"
    ) %>%
    group_by(name) %>%
    summarise(popularity = sum(n)) %>%
    arrange(desc(popularity)) %>%
    head(n = 10)
```

```
## # A tibble: 10 x 2
##    name     popularity
##    <chr>         <int>
##  1 George      1464186
##  2 Joshua      1202454
##  3 Jose         560679
##  4 Kyle         477768
```

```
##   5 Lawrence     456773
##   6 Joe          450780
##   7 Willie       448702
##   8 Jesse        416530
##   9 Bruce        382257
##  10 Eugene       378539
```

```r
# B

babynames::babynames %>%
    filter(
        stringr::str_detect(string = name, pattern = "(joe|jo|Joe|Jo)$")
    ) %>%
    group_by(name) %>%
    summarise(popularity = sum(n)) %>%
    arrange(desc(popularity)) %>%
    head(n = 10)
```

```
## # A tibble: 10 x 2
##    name      popularity
##    <chr>          <int>
##  1 Joe           462099
##  2 Jo            180579
##  3 Maryjo          7017
##  4 Billiejo        1455
##  5 Marijo          1280
##  6 Bobbijo         1237
##  7 Bobbiejo        1009
##  8 Alejo            794
##  9 Bettyjo          764
## 10 Amyjo            486
```