logistic_regression

2022-07-31

2) Complete data model is logistic regression & probability to be missing depends ## only on Y

WTS: If probability to be missing depends ONLY on Y and NOT on X, and missing

data is confined to either Y or X (will confine to just X in our case), then

LD is more efficient (and generates unbiased regression coefficients)

```
create.data <- function(beta_0 = 0, beta_1 = 1, beta_2 = 2, n = 200){
x1 = rnorm(n)             # some continuous variables
x2 = rnorm(n)
z = beta_0 + beta_1*x1 + beta_2*x2        # Here, B_0 = 1, B_1 = 2.
pr = 1/(1+exp(-z))          # pass through an inv-logit (sigmoid) function to
                            # to constrain to [0,1];
Y = rbinom(n,1,pr)
data = as.data.frame(cbind(Y, x1, x2))

}
```

```
data <- create.data()
data
```

relative to multiple imputation.

```
##    Y           x1          x2
## 1  0 -0.966659885 -2.30957503
## 2  0  0.413768780 -0.35886034
## 3  1  0.625555293  0.91329839
## 4  1 -0.634437081  0.77801411
## 5  1 -0.411279070  0.17687392
## 6  1  0.653376668  0.04989277
## 7  0 -0.302163166  0.84027267
## 8  0 -0.961532540  0.57986432
## 9  1 -0.618665710  0.19497418
## 10 0  0.371382321 -0.27025280
## 11 0 -0.247532764  0.21515521
## 12 0 -0.336329901 -1.54203475
## 13 0  0.004056327 -1.20576957
## 14 1  0.501081401 -0.08893358
## 15 0 -0.705540562  0.81210662
## 16 0 -0.615832638 -1.49450713
## 17 0 -0.184008211  0.42901230
## 18 0  0.837940652  0.12833079
```

```
## 19  0  0.819690630 -1.65803739
## 20  1  1.193642780  0.45329521
## 21  0  0.131278195 -0.07680205
## 22  1  0.805655229  0.99438741
## 23  1  0.712021082 -0.45203549
## 24  0 -1.668358308 -0.73304524
## 25  1 -0.059996743 -0.17020636
## 26  1 -0.205635252  1.99482164
## 27  1  0.836106896  0.88067690
## 28  0  0.489556921 -0.87502948
## 29  0  0.081252159  0.62764734
## 30  0  0.573302732 -0.90715594
## 31  0  0.396522310 -0.97039127
## 32  0 -0.526714514 -1.43092381
## 33  0 -1.489086229 -1.04769958
## 34  1  0.199621396  0.72281435
## 35  1 -1.318658086  0.41581700
## 36  0  1.383108592 -2.26498937
## 37  0 -1.943089866  0.07178334
## 38  1  0.099457934  2.35648059
## 39  0 -1.324434346 -0.36506795
## 40  1 -0.226716268  1.57156326
## 41  1  1.431418131  0.62070354
## 42  1 -0.994008415  0.91063067
## 43  1  1.120606779  1.43643495
## 44  0 -0.562196836 -0.69743081
## 45  1 -0.050548584 -0.40461823
## 46  0 -1.550956583 -1.79793843
## 47  1 -0.174718664  0.19612777
## 48  1 -0.859068201  1.24072430
## 49  0 -0.587036200 -0.32485174
## 50  0  0.330046965 -0.44403283
## 51  1  0.760163954  1.22764755
## 52  1 -0.242811734 -0.01933909
## 53  1 -0.202576667  1.26579044
## 54  0 -0.659421534  0.34978793
## 55  1  0.539414626 -0.18295850
## 56  0 -0.103844212  0.21223964
## 57  1 -0.070973340  0.93250886
## 58  1  0.608006967  0.61739200
## 59  1  0.816535792 -0.02100856
## 60  0  0.313248555  0.35740752
## 61  1 -0.842362405  1.15403657
## 62  1 -1.179652440  2.41894796
## 63  1  1.711935199  1.01551253
## 64  0 -0.995255839  0.80895930
## 65  1 -0.552176050  0.53940675
## 66  0 -0.937066644 -0.19451250
## 67  1  1.044179939  0.18922335
## 68  0  0.225420362 -0.61670594
## 69  1  1.357948148  1.31805898
## 70  1 -0.093994899  0.01796951
## 71  0  0.871702776 -0.96894692
## 72  1 -0.980833465  0.98703436
```

```
## 73  1  0.507160051  0.10652636
## 74  0  0.099785200 -1.04094519
## 75  0 -0.901693333  0.20505689
## 76  0 -0.474255501  0.81822973
## 77  1 -1.043705425  0.99069077
## 78  1 -0.066003670 -0.05117990
## 79  1  0.121017966  0.58687541
## 80  1 -0.927557854  0.82505239
## 81  1  1.882859880  0.14422974
## 82  0 -0.626589089  0.11977845
## 83  1  0.204913693  0.23228225
## 84  1  1.399639217  0.83592291
## 85  1  0.610669709  1.28622751
## 86  1 -1.032985271  0.84659221
## 87  1  0.610046615  0.49394264
## 88  1 -0.372750275  0.45289534
## 89  0 -1.852423425 -0.73920195
## 90  1  0.886502879  0.74559819
## 91  1 -1.605507687  0.26776339
## 92  1  1.430078936 -1.43723950
## 93  1 -0.825435322  0.57927628
## 94  0 -0.474063027 -0.64280691
## 95  1  0.364598734  0.44762296
## 96  1 -2.068827387  1.15646894
## 97  0  0.604965343 -0.10149201
## 98  1  1.664206419  1.17033305
## 99  0  3.006618174 -0.09340858
## 100 0  0.198483401 -0.13611669
## 101 0 -0.643154225 -0.91269666
## 102 1  0.380278493 -0.28972030
## 103 1 -0.342182660  1.66679690
## 104 1 -1.022550555  0.54557690
## 105 0 -0.701967158 -1.12266377
## 106 1  0.126428672  0.63422318
## 107 0 -0.754764707 -0.51448471
## 108 1  0.365500539  0.81788112
## 109 0  0.059043981  0.36176285
## 110 0 -1.319651434 -0.65364493
## 111 0 -1.003563365  0.25706587
## 112 1 -0.660043247 -0.96055065
## 113 1 -0.362333178  2.29510606
## 114 1  1.137560612  1.15000704
## 115 1  1.258399090  0.59349911
## 116 1  1.505411120  2.15324517
## 117 1  1.022761137  1.10726405
## 118 0 -1.825587532 -0.04005304
## 119 1  0.245271007  0.12504483
## 120 0 -0.980823074 -0.01020598
## 121 0  0.986570794 -0.63860904
## 122 0  0.237813573 -2.05281443
## 123 0 -1.144811313 -1.36318111
## 124 1  0.212885586  0.52330532
## 125 0  0.987571627  0.01222115
## 126 1 -1.394156827  1.94532269
```

```
## 127 1  1.022251100  0.76855727
## 128 1  0.024525765  2.05479104
## 129 1 -0.186578030  0.06152101
## 130 0 -1.001470233  0.12837315
## 131 0 -0.785422908 -0.97552050
## 132 0  0.196825432 -0.85247983
## 133 1 -0.054953309 -0.69265294
## 134 0  2.077601949 -1.81831686
## 135 1  1.426131206 -0.21936592
## 136 0 -0.555480101 -0.34634251
## 137 1  1.944718884 -0.60826573
## 138 1  0.979450033 -0.47398293
## 139 1  1.871309322 -1.07512152
## 140 0  1.266717348 -0.88023290
## 141 1 -2.084122862  0.89738962
## 142 0  0.540317118  0.12602731
## 143 0 -1.722405449 -0.38593160
## 144 1 -0.183612906  0.14154569
## 145 0 -1.549367430  1.25234242
## 146 1  0.839690457  1.38309788
## 147 1 -1.115205057  0.77043387
## 148 0 -0.624486404 -2.06144113
## 149 1  1.634785826  0.39421843
## 150 0 -0.593964265  0.21605605
## 151 1 -0.365565264  0.44858699
## 152 0 -1.834391845  0.69863639
## 153 0 -0.689952058  0.24297540
## 154 0 -1.152356629 -2.09228231
## 155 0  0.108724563  1.07316920
## 156 0 -0.864380572 -0.65370281
## 157 0 -1.709152705  0.88971305
## 158 0 -1.647813901 -0.93443328
## 159 1  0.989625220  0.73508761
## 160 1  0.454098344  1.27747736
## 161 1  1.528591966 -0.40259252
## 162 1  0.038031337  1.75088627
## 163 0 -0.133959651  1.64630400
## 164 1  0.403917167 -0.06826208
## 165 1  0.702442407 -0.61543383
## 166 0  0.583063061  0.46794728
## 167 1 -0.599556423  1.06156384
## 168 0 -0.332641845  0.12696376
## 169 0  0.103207262  1.17420495
## 170 1  1.899675543  0.01472493
## 171 0  0.624631634 -0.86251490
## 172 1  0.501530110  0.07305673
## 173 0 -2.086114468  0.38383323
## 174 0 -0.912478380  0.71835592
## 175 1 -0.824903866  1.70229069
## 176 0  0.327485867 -0.92420976
## 177 0 -0.726021208 -0.63387890
## 178 0 -0.382031725 -0.30999654
## 179 1  0.421218814  0.16144099
## 180 1 -0.982807465  0.25897670
```

```
## 181 0 -1.098778333 -0.11839672
## 182 0 -1.359567301 -0.37002376
## 183 1 -0.040439502  0.15147097
## 184 0  0.270764019 -0.81552772
## 185 1 -0.622974273 -0.32375970
## 186 0 -0.967619896 -1.25477543
## 187 0 -1.830583436 -0.60669061
## 188 0 -0.172838444 -1.25541822
## 189 0 -0.916393519  1.51207787
## 190 1 -0.382614988  0.52010339
## 191 1 -0.600311480 -0.16327685
## 192 0  0.809542254 -0.22956870
## 193 0 -1.240968437 -0.66415987
## 194 0  0.398639061 -0.37865741
## 195 1  0.284672138  0.14290382
## 196 0 -0.849677920  0.62723631
## 197 0 -1.996663105  0.81319568
## 198 1  0.485311316  0.38141688
## 199 1 -0.290281048  2.48959665
## 200 0 -1.001175429 -0.58261810
```

Add Missingness

```
# probability of missingness is greater for x1/x2 if Y == 1.
MNAR.make.missing <- function(data, prob_missing_larger = 0.2,
                              prob_missing_smaller = 0.6){
  # Setting up the randomness categories for missingness in x1, x2,
  higher <- data %>% filter(Y == 1) %>% select(x1)
  rx1_larger <- rbinom(nrow(higher), 1, prob_missing_larger)
  rx1_smaller <- rbinom(nrow(data) - nrow(higher), 1, prob_missing_smaller)
  rx2_larger <- rbinom(nrow(higher), 1, prob_missing_larger)
  rx2_smaller <- rbinom(nrow(data) - nrow(higher), 1, prob_missing_smaller)
  rx1 <- c(rx1_larger, rx1_smaller)
  rx2 <- c(rx2_larger, rx2_smaller)
  data <- data %>%
    arrange(desc(Y)) %>%
    cbind(rx1, rx2)
  # Implementing the missingness in x1, X_3, X_4
  data <- data %>% mutate(x1 = case_when(rx1 == 1 ~ as.numeric(NA),
                    rx1 == 0 ~ as.numeric(data$x1))) %>%
    mutate(x2 = case_when(rx2 == 1 ~ as.numeric(NA),
                    rx2 == 0 ~ as.numeric(data$x2)))
  # Remove setup variables
  data <- select(data, -c(rx1, rx2))
  data
}
```

Multiple Imputation

```
# Simulate multiple imputation, obtaining estimates and 95% confidence interval.
simulate_MI2 <- function(runs = 100) {
  res <- array(NA, dim = c(3, runs, 3))
```

```r
  times <- array(NA, dim = c(runs, 1, 1))
  dimnames(res) <- list(c("Intercept", "x1", "x2"),
                        as.character(1:runs), c("estimate", "2.5%", "97.5%"))
  sim_dataset <- as.data.frame(create.data(n = 1000))
  for (run in 1:runs){
      # Note that time is only measured for the MI/imp steps
      # (i.e. filtering, predicting)
    missingness_sim_dataset <- MNAR.make.missing(sim_dataset, 0.2, 0.8)
    start_time <- Sys.time()
    imp_MI <- mice(missingness_sim_dataset, print = FALSE)
    fit <- with(imp_MI, glm(Y ~ x1 + x2, family = "binomial"))
    end_time <- Sys.time()
    tab <- summary(pool(fit), "all", conf.int = TRUE)
    res[1, run, ] <- as.numeric(tab[1, c("estimate", "2.5 %", "97.5 %")])
    res[2, run, ] <- as.numeric(tab[2, c("estimate", "2.5 %", "97.5 %")])
    res[3, run, ] <- as.numeric(tab[3, c("estimate", "2.5 %", "97.5 %")])

    times[run, 1, 1] <- as.numeric(end_time - start_time)
  }
  list(res, times)
}
```

```r
# Run 100 iterations
res_MI2 <- simulate_MI2(100)
```

```r
# Obtain confidence intervals & estimates for all coefficients, intercept.
apply(res_MI2[[1]], c(1, 3), mean, na.rm = TRUE)
```

```
##            estimate       2.5%     97.5%
## Intercept 0.1002261 -0.4251241 0.6255763
## x1        1.1277327  0.1873987 2.0680666
## x2        2.2181855  1.0932241 3.3431469
```

```r
# Mean time for the multiple imputation instances
times <- res_MI2[[2]]
mean(times)
```

```
## [1] 0.2416485
```

```r
Multiple_Imputation <- mean(times)
```

```r
# Evaluating imputation method performance for estimating
# all parameters of interest.
res <- res_MI2[[1]]
true <- c(0, 1, 2)
Raw_Bias <- rowMeans(res[,, "estimate"]) - true
Percent_Bias <- 100 * abs((rowMeans(res[,, "estimate"]) - true)/ true)
Coverage_Rate <- rowMeans(res[,, "2.5%"] < true & true < res[,, "97.5%"])
Average_Width <- rowMeans(res[,, "97.5%"] - res[,, "2.5%"])
RMSE <- sqrt(rowMeans((res[,, "estimate"] - true)^2))
MI_measures <- data.frame(Raw_Bias, Percent_Bias, Coverage_Rate, Average_Width, RMSE)
knitr::kable(round(MI_measures, 3), align = "ccccc") %>% kable_styling()
```

	Raw_Bias	Percent_Bias	Coverage_Rate	Average_Width	RMSE
Intercept	0.100	Inf	1.00	1.051	0.170
x1	0.128	12.773	0.98	1.881	0.309
x2	0.218	10.909	0.99	2.250	0.360

Listwise Deletion

```
# Simulate listwise deletion, obtaining estimates and 95% confidence interval.

simulate_LD <- function(runs = 100){
  res <- array(NA, dim = c(3, runs, 3))
  dimnames(res) <- list(c("Intercept", "x1", "x2"),
                        as.character(1:runs), c("estimate", "2.5%", "97.5%"))
  times <- array(NA, dim = c(runs, 1, 1))
  sim_dataset <- as.data.frame(create.data(n = 1000))
  # Note that time is only measured for the LD/imp steps (i.e. filtering, predicting)
  for (run in 1:runs){
    missingness_sim_dataset <- MNAR.make.missing(sim_dataset, 0.2, 0.5)
    start_time <- Sys.time()
    filtered_sim_dataset <- missingness_sim_dataset %>%
      select(Y, x1, x2) %>%
      filter(!is.na(x1), !is.na(x2))

    fit <- with(filtered_sim_dataset, glm(Y ~ x1 + x2, family = "binomial"))
    end_time <- Sys.time()
    times[run, 1, 1] <- as.numeric(end_time - start_time)
    # loop over each variable. Note we do the imputation just ONCE b/c LD is
    # deterministic.
    for (var in 1:3){
      edges <- as.numeric((confint.default(fit))[var,])
      estimate <- as.numeric(fit$coefficients)[var]
      interval <- c(estimate, edges)
      res[var, run, ] <- interval
      }

  }
  list(res, times)
}
```

```
result_LD <- simulate_LD(100)
```

```
# Obtain confidence intervals & estimates for all coefficients, intercept.
apply(result_LD[[1]], c(1, 3), mean, na.rm = TRUE)
```

```
##            estimate      2.5%    97.5%
## Intercept 0.9067962 0.6193571 1.194235
## x1        1.1459560 0.8061650 1.485747
## x2        2.2936344 1.8077398 2.779529
```

```
# Evaluating imputation method performance for estimating
# all parameters of interest.
res <- result_LD[[1]]
true <- c(0, 1, 2)
Raw_Bias <- rowMeans(res[,, "estimate"]) - true
```

	Raw_Bias	Percent_Bias	Coverage_Rate	Average_Width	RMSE
Intercept	0.907	Inf	0.00	0.575	0.913
x1	0.146	14.596	0.93	0.680	0.208
x2	0.294	14.682	0.89	0.972	0.361

	average-runtime
Multiple_Imputation	0.2416485
ListwiseDeletion	0.0073440

```
Percent_Bias <- 100 * abs((rowMeans(res[,, "estimate"]) - true)/ true)
Coverage_Rate <- rowMeans(res[,, "2.5%"] < true & true < res[,, "97.5%"])
Average_Width <- rowMeans(res[,, "97.5%"] - res[,, "2.5%"])
RMSE <- sqrt(rowMeans((res[,, "estimate"] - true)^2))
LD_measures <- data.frame(Raw_Bias, Percent_Bias, Coverage_Rate, Average_Width, RMSE)
knitr::kable(round(LD_measures, 3), align = "ccccc") %>% kable_styling()
```

```
# Mean time for 100 instances of LD
times_LD <- result_LD[[2]]
ListwiseDeletion <- mean(times_LD)
```

```
mean_times <- as.data.frame(rbind(Multiple_Imputation, ListwiseDeletion), col.names = "average_runtime")
colnames(mean_times) <- "average-runtime"
mean_times
```

```
##                     average-runtime
## Multiple_Imputation     0.241648512
## ListwiseDeletion        0.007343969
```

```
knitr::kable(mean_times) %>% kable_styling(full_width = F)
```