# Utilizing a Capture-Recapture Strategy to Accelerate Infectious Disease Surveillance

This page aims to illustrate the Data Generation, Analysis, Simulation Study as well as the Numerical Example for the manuscript **"Utilizing a Capture-Recapture Strategy to Accelerate Infectious Disease Surveillance"**.

All functions are available in this Github repository and can be loaded as follows.

```
source("FUN_CRC_project_SeSp.R")
```

### 1. Data Generation

Suppose we generate a dataset with total population size $N_{total} = 1,000$ and true disease prevalence $p = 0.1$. Data were generated in such a way that among those with disease, 50% of individuals exhibited symptoms. In contrast, only 10% of those without disease showed symptoms. The Stream 1 sample was drawn to reflect voluntary-based non-representative surveillance data, selecting 80% of individuals with symptoms for testing as opposed to 10% of those without symptoms. Stream 2 was generated as the anchor stream independently of Stream 1, with the sample size $n_2 = 50$. We then consider "low" level of misclassification (e.g., $Se_1 = Sp_1 = 0.9$, $Se_1 = Sp_1 = 0.95$). This is the same setting as Table 2 in the paper.

Therefore we set the following parameters:

```
N = 1000

p_case = 0.1
p_case_sym = 0.5
p_not_case_sym = 0.1

N_true = c(rep(1,round(N*p_case)),rep(0,round(N*(1-p_case))))
Npos = sum(N_true)
Nneg = N-Npos

# two-steam surveillance
p_test1_givSym = .8
p_test1_givAsym = .1
Se1 = 0.9
Sp1 = 0.9

n2 = 50
p_stream2 = n2/N
p_test2_givSym = p_stream2
p_test2_givAsym = p_stream2
Se2 = 0.95
Sp2 = 0.95
```

and we can generate the dataset (including testing results in Stream 1 and 2, as well as `n1-n9`) by the following functions.

```
set.seed(1111111)
# generate the symptom status
N_sym = rep(0,N)
N_sym[which(N_true == 1)] = rbinom(Npos,1,p_case_sym)
N_sym[which(N_true == 0)] = rbinom(N-Npos,1,p_not_case_sym)
p_sym = sum(N_sym)/N

# generate steam 1
simu1 = simu_sym3(N,N_sym,p_sym,N_true,p_test1_givSym,p_test1_givAsym,Se1,Sp1)
test1 = simu1$test
testpos1 = simu1$testpos

# generate steam 2
simu2 = simu_sym_RS(N,N_true,p_stream2,Se2,Sp2)
test2 = simu2$test
testpos2 = simu2$testpos

obs_summary = two_by_two_table2(test1,testpos1,test2,testpos2)
n1 = obs_summary$n1
n2 = obs_summary$n2
n3 = obs_summary$n3
n4 = obs_summary$n4
n5 = obs_summary$n5
n6 = obs_summary$n6
n7 = obs_summary$n7
n8 = obs_summary$n8
n9 = obs_summary$n9
```

### 2. Data Analysis

There are two estimators compared in Section 2 of the paper: Random Sampling (RS) based estimator and Capture-Recapture (CRC) estimator. We use function `RS_mis` for RS estimator in eqn. (1)-(3), and `ML_est_SESP_numerical` or `ML_est_SESP_closedform` for CRC estimator in eqn. (5)-(8).

As an example, we use the following programs to calculate RS estimator:

```
RS = RS_mis(N, n_stream2=n1+n2+n3+n4+n7+n8, n_pos_stream2=n1+n4+n7, Se2=Se2, Sp2=Sp2)
N_RS = RS$Nhat
N_RS_sd = RS$SEhat
RS = data.frame(est=N_RS, se=N_RS_sd, pct_025=N_RS-1.96*N_RS_sd, pct_975=N_RS+1.96*N_RS_sd,
           width=1.96*N_RS_sd*2)
print(RS,row.names=FALSE)
```

```
##       est       se  pct_025  pct_975    width
##  188.8889 64.54436 62.38195 315.3958 253.0139
```

and we calculate the closed form CRC estimator as follows:

```
MLE = ML_est_SESP_closedform(n1,n2,n3,n4,n5,n6,n7,n8,n9,Se1_Sp1_par=c(Se1,Sp1),
                             Se2_Sp2_par=c(Se2,Sp2))
N_SESP = MLE$mle
N_SESP_sd = MLE$mle_se
SESP =data.frame(est=N_SESP, se=N_SESP_sd, pct_025=N_SESP-1.96*N_SESP_sd,
                 pct_975=N_SESP+1.96*N_SESP_sd,width=1.96*N_SESP_sd*2)
print(SESP,row.names=FALSE)
```

```
##       est       se  pct_025  pct_975    width
##  125.5507 50.72714 26.12552 224.9759 198.8504
```

Except for the Wald-type confidence interval for CRC estimator, we also provide a Bayesian credible interval in Section 2.4.

```
BC_interval = BC_interval_SESP(n1,n2,n3,n4,n5,n6,n7,n8,n9,Se1,Sp1,Se2,Sp2)
BC_interval2 = RS_BC2(N, n_pos_stream2=n1+n4+n7,n_stream2=n1+n2+n3+n4+n7+n8,Se2,Sp2)
if(BC_interval$BC_width>(N * BC_interval2$BC_width)) {
  lower = N * BC_interval2$BC_lower
  upper = N * BC_interval2$BC_upper
  BC_interval = list(BC_lower = lower,BC_upper = upper,BC_width = upper - lower)
}

if (BC_interval$BC_upper >= Npos &&
    BC_interval$BC_lower <= Npos) {
  BC_interval_coverage = 1
} else{
  BC_interval_coverage = 0
}
BC_interval_width = BC_interval$BC_width

SESP_bc =data.frame(est=N_SESP, se=N_SESP_sd, pct_025=BC_interval$BC_lower,
                    pct_975=BC_interval$BC_upper,width=BC_interval_width)
print(SESP_bc,row.names=FALSE)
```

```
##       est       se  pct_025 pct_975    width
##  125.5507 50.72714 53.82643 251.276 197.4496
```

Meanwhile, we also proposed a MI-based approach when Se and Sp are estimated from the validation data. More details are available in “Numeric Example” below.

## 3. Simulation Study

To perform simulation study, we use the following loop to evaluate the simulation results and generate Table 2-4.

```
tmp = NULL
for(p_case in seq(0.1,0.5,0.2)){
  for(n2_samples in c(50, 100, 300) ){
     p_stream2 = n2_samples/N
     res = find_p_p2(p_case,p_stream2, N, Se1,Sp1=Se1,Se2=Se1+0.05,Sp2=Se1+0.05)
     tmp = rbind(tmp,t(c(p_case,p_stream2,res$res.true,res$res.RS[1:5],res$res.SESP[1:5],
                         res$res.BC_width, res$res.BC_pct,
                         res$res.SESP2, res$res.SESP2_sd, Se1)))
   }
}
colnames(tmp) = c('p_case','p_stream2','N_true','RS.mean','RS.sd','RS.avgse','RS.width',
                  'RS.Clpct','SESP.mean','SESP.sd','SESP.avgse','SESP.width','SESP.Clpct',
                  'SESP.BCwidth','SESP.BCpct','SESP2.mean','SESP2.sd', 'Se1')
(result = round(tmp[,],2))
```

```
##       p_case p_stream2 N_true RS.mean RS.sd RS.avgse RS.width RS.Clpct
##  [1,]    0.1      0.05    100   98.22 54.28    52.74   206.76     91.6
##  [2,]    0.1      0.10    100  101.98 37.31    37.39   146.59     94.7
```

```
## [3,]    0.1      0.30    100   99.09 20.94     20.07     78.69      93.4
## [4,]    0.3      0.05    300  298.93 72.41     71.70    281.05      93.2
## [5,]    0.3      0.10    300  297.88 50.73     49.66    194.66      94.9
## [6,]    0.3      0.30    300  299.00 25.87     26.17    102.58      95.7
## [7,]    0.5      0.05    500  502.33 76.56     76.99    301.80      94.2
## [8,]    0.5      0.10    500  499.14 52.15     53.29    208.89      96.5
## [9,]    0.5      0.30    500  500.16 27.31     27.92    109.44      95.1
##      SESP.mean SESP.sd SESP.avgse SESP.width SESP.CIpct SESP.BCwidth
## [1,]     99.37   41.69      42.92     168.24       93.0       157.59
## [2,]    101.93   30.92      30.98     121.44       93.2       117.60
## [3,]     99.74   17.55      17.66      69.21       94.6        69.16
## [4,]    299.20   58.86      57.66     226.03       92.4       220.54
## [5,]    298.18   40.48      40.32     158.04       94.1       156.34
## [6,]    299.63   21.56      22.38      87.72       95.5        88.31
## [7,]    502.08   64.90      63.36     248.36       92.7       240.00
## [8,]    500.15   42.52      44.15     173.08       95.0       170.49
## [9,]    499.38   23.75      24.18      94.79       94.8        95.36
##      SESP.BCpct SESP2.mean SESP2.sd Se1
## [1,]       94.6      99.22    41.77 0.9
## [2,]       94.5     101.91    30.85 0.9
## [3,]       95.4      99.62    17.47 0.9
## [4,]       95.0     299.19    58.85 0.9
## [5,]       95.0     298.19    40.47 0.9
## [6,]       95.7     299.63    21.42 0.9
## [7,]       94.6     502.09    64.89 0.9
## [8,]       95.5     500.11    42.50 0.9
## [9,]       95.2     499.44    23.65 0.9
```

### 4. Numeric Example

Now we analyze the numerical example provided in this paper. The data can be found in (`data_numerical_example.R`).

```r
library(pander)
set.seed(111111)
source("data_numerical_example.R")
```

The data includes nine cell counts (`n1`-`n9`):

```r
data_obs = c(n1,n2,n3,n4,n5,n6,n7,n8,n9)
data_obs
```

```
## [1]   3  12   0   2  27 130   6  77 743
```

as well as the 2-by-2 table of testing results from Table 5 of paper: True validation data from Murakami et al. (2023),

```r
n_SE1SP1_validation = c(n11_t1,n10_t1,n01_t1,n00_t1)
matrix(n_SE1SP1_validation,2,2,byrow = T)
```

```
##      [,1] [,2]
## [1,]   65    1
## [2,]   38  552
```

```r
Se1_initial = n11_t1/(n11_t1+n01_t1)
Sp1_initial = n00_t1/(n10_t1+n00_t1)
c(Se1_initial,Sp1_initial)
```

```
## [1] 0.6310680 0.9981917
```

and from Casati et al. (2022),

```
n_SE2SP2_validation = c(n11_t2,n10_t2,n01_t2,n00_t2)
matrix(n_SE2SP2_validation,2,2,byrow = T)
```

```
##      [,1] [,2]
## [1,]   89    0
## [2,]    6  100
```

```
Se2_initial = n11_t2/(n11_t2+n01_t2)
Sp2_initial = n00_t2/(n10_t2+n00_t2)
c(Se2_initial,Sp2_initial)
```

```
## [1] 0.9368421 1.0000000
```

**3.1 Proposed Approach for Reliable Se, Sp Parameters** In Section 4, we first analyze the data assuming that the misclassification parameters of each data streams are known. Then we can directly calculate the case count estimates and corresponding confidence (or credible) intervals using the approaches introduced in Section 2. That is,

```
RS = RS_mis(N=sum(data_obs), n_stream2=n1+n2+n3+n4+n7+n8,
            n_pos_stream2=n1+n4+n7, Se2=Se2_initial, Sp2=Sp2_initial)
N_RS = RS$Nhat
N_RS_sd = RS$SEhat
RS_m1 = data.frame(est=N_RS, se=N_RS_sd, pct_025=N_RS-1.96*N_RS_sd,
                   pct_975=N_RS+1.96*N_RS_sd, width=1.96*N_RS_sd*2)
print(RS_m1,row.names=FALSE)
```

```
##       est       se  pct_025  pct_975   width
##  117.4157 31.96812 54.75822 180.0732 125.315
```

and CRC estimate results:

```
MLE = ML_est_SESP_closedform(n1,n2,n3,n4,n5,n6,n7,n8,n9,Se1_Sp1_par=c(Se1_initial,Sp1_initial),
                             Se2_Sp2_par=c(Se2_initial,Sp2_initial))
N_SESP = MLE$mle
N_SESP_sd = MLE$mle_se
SESP_m1 =data.frame(est=N_SESP, se=N_SESP_sd, pct_025=N_SESP-1.96*N_SESP_sd,
                    pct_975=N_SESP+1.96*N_SESP_sd, width=1.96*N_SESP_sd*2)
print(SESP_m1,row.names=FALSE)
```

```
##       est       se  pct_025  pct_975    width
##  111.5433 24.67128 63.18762 159.8991 96.71143
```

as well as CRC estimate with Bayesian credible interval:

```
tmp2 = BC_interval_SESP(n1,n2,n3,n4,n5,n6,n7,n8,n9,Se1_initial,Sp1_initial,
                        Se2_initial,Sp2_initial)
SESP_m1_bc = data.frame(est=N_SESP, se=N_SESP_sd, pct_025=tmp2$BC_lower,
                        pct_975=tmp2$BC_upper,width=tmp2$BC_width)
print(SESP_m1_bc,row.names=FALSE)
```

```
##       est       se  pct_025  pct_975    width
##  111.5433 24.67128 75.07522 172.5666 97.49134
```

**3.2 MI-based Approach for Estimable Se, Sp Parameters** With access to validation data for estimating the misclassification parameters as in Table 5, we recommend the approach introduced in Section

2.5 that adapts the multiple imputation (MI) paradigm (Rubin, 1987) to account for uncertainty in these parameters. In this repository, we define a function `MI_main` to implement MI to calculate each estimators introduced in the paper. The results are as follows.

```
tmp = unlist(MI_main(data_obs,n_SE1SP1_validation,n_SE2SP2_validation))
RS_m2 = tmp[1:5]
SESP_m2 = tmp[6:10]
SESP_m2_bc = tmp[c(6,7,11:13)]
RS_m2
```

```
##        RS_MI     RS_MI.se  RS_MI_lower  RS_MI_upper RS_MI_length
##    113.65881     33.30964     48.37191    178.94571    130.57380
```

```
SESP_m2
```

```
##        SESP_MI     SESP_MI.se  SESP_MI_lower  SESP_MI_upper SESP_MI_length
##      108.20869       26.04108       57.16818      159.24920      102.08102
```

```
SESP_m2_bc
```

```
##             SESP_MI           SESP_MI.se   SESP_BC_lower.2.5%
##           108.20869             26.04108             68.45380
##  SESP_BC_upper.97.5% SESP_BC_length.97.5%
##           172.67935            104.22555
```

**3.3 Numerical Example Results** Overall, we can combine all results and compare each methods as follows (same as Table 6 in the paper).

| | est | se | 2.5% | 97.5% | width |
|---|---|---|---|---|---|
| **RS_m1** | 117.4 | 32 | 54.8 | 180.1 | 125.3 |
| **SESP_m1** | 111.5 | 24.7 | 63.2 | 159.9 | 96.7 |
| **SESP_m1_bc** | 111.5 | 24.7 | 75.1 | 172.6 | 97.5 |
| **RS_m2** | 113.7 | 33.3 | 48.4 | 178.9 | 130.6 |
| **SESP_m2** | 108.2 | 26 | 57.2 | 159.2 | 102.1 |
| **SESP_m2_bc** | 108.2 | 26 | 68.5 | 172.7 | 104.2 |