Integrating Probability and Nonprobability Samples for Survey Inference

Arkadiusz Wiśniowski[*], Joseph W. Sakshaug[†], Diego A. Perez Ruiz[‡], Annelies G. Blom[§]

Last updated: 09 December 2020

Introduction

This document introduces computer code that reproduces the results of estimating model parameters in a linear regression using Bayesian inference. The estimation uses simulated probability data with prior distributions constructed using simulated nonprobability data. It is based on the method as proposed in:

Integrating Probability and Nonprobability Samples for Survey Inference, *Journal of Survey Statistics and Methodology*, Volume 8, Issue 1, February 2020, Pages 120–147, https://doi.org/10.1093/jssam/smz051.

The repository with the code is available at https://github.com/a-wis/integrating_prob_nonprob.

Code

Required packages

```
library(readr)
library(tidyverse)
```

Generating toy data

The function `gen.sample()` generates a sample of size `n` of a continuous response variable with the standard deviation `sd_response`. The mean is a linear equation given by parameters `beta` (a vector including intercept) and two continuous predictors with means provided in the vector `m_x` and standard deviations `sd_cov1` and `sd_cov2`, respectively. The two predictors can also be correlated with correlation coefficient `corr`. The response can be generated assuming a certain level of `bias` in the second predictor (for generating a nonprobability sample). Argument `bias` can also take a vector as an input.

```
gen.sample = function(beta = c(1, 0.5, 0.1), n = 50, m_x = c(0, 5), sd_response = 1,
    sd_cov1 = 1, sd_cov2 = 1, corr = 0.1, bias = 1) {
    x1 = rnorm(n, m_x[1], sd_cov1)  #covariate 1
    x2_m = m_x[2]  # covariate2 mean
    x2 = x2_m + sd_cov2/sd_cov1 * corr * (x1 - m_x[1]) + rnorm(n, 0, sqrt(1 - corr^2) *
        sd_cov2)
    X = matrix(c(rep(1, n), x1, x2), n, length(beta))
```

[*] Social Statistics Department, University of Manchester, United Kindgom; Contact: a.wisniowski@manchester.ac.uk

[†] Department of Statistical Methods Research, Institute for Employment Research; Department of Statistics, Ludwig Maximilian University of Munich; and the School of Social Sciences, University of Mannheim, Germany

[‡] Department of Mathematics, University of Manchester, United Kindgom

[§] Collaborative Research Center 884 “Political Economy of Reforms” and the School of Social Sciences, University of Mannheim, Germany

```
    # covariate No. 2 with correlation with cov1 if bias is specified as a vector,
    # response is calculated for a given set of sampled X
    if (length(bias) == 1) {
        Y = X %*% (beta * bias) + rnorm(n, 0, sd_response)
    } else {
        Y = X %*% c(beta[-3], beta[3] * bias[1]) + rnorm(n, 0, sd_response)
        for (i in 2:length(bias)) {
            Y = cbind(Y, X %*% c(beta[-3], beta[3] * bias[i]) + rnorm(n, 0, sd_response))
        }
    }
    # returns a list with response Y and predictors matrix X as an output
    return(list(Y = Y, X = X))
}
```

Calculating posterior distribution

This function calculates posterior mean and variance of the vector of linear model coefficients and mean and variance of the precision (inverse variance) of the linear regression model.

```
calc.posterior = function(mu_0, k_0, Vin = diag(length(mu_0)), a_0 = 0, b_0 = 0,
    y = Y, x = X) {
    if (dim(as.matrix(x))[2] != length(mu_0) | length(y) != dim(as.matrix(x))[1])
        print("Dimensions mismatch!") else {
        n_mu_0 = length(mu_0)
        p = dim(as.matrix(x))[2]
        n = length(y)
        V = Vin * k_0
        # some OLS values
        mu_hat = as.matrix(lm(y ~ x - 1)$coefficients)
        xtx = t(x) %*% x
        RSS = t(y - x %*% mu_hat) %*% (y - x %*% mu_hat)
        # posterior mean of mu
        Sigma_t = solve(xtx + solve(V))
        W = Sigma_t %*% xtx
        mu_mean = W %*% mu_hat + (diag(p) - W) %*% as.matrix(mu_0)

        SS = RSS + t(mu_hat - as.matrix(mu_0)) %*% solve(solve(xtx) + V) %*% (mu_hat -
            as.matrix(mu_0))
        v = n + 2 * a_0
        mu_var = Sigma_t * (as.numeric(SS) + 2 * b_0)/(n + 2 * a_0 - 2)
        # posterior of tau
        tau_mean = (n/2 + a_0)/(SS/2 + b_0)
        tau_var = (n/2 + a_0)/(SS/2 + b_0)^2
        return(list(mu_mean = mu_mean, mu_cov = sqrt(diag(mu_var)), tau_mean_sd = c(tau_mean,
            sqrt(tau_var))))
    }
}
```

Priors functions

These functions represent the assumptions of the prior distributions constructed using nonprobability data (Section 2.1 of the article).

```r
# Conjugate, Eq. 14
fun.hot.c<-function(hot.n,n){if (hot.n < 0.05) 1/log(n) else 1/n}
# conjugate-distance, Eq. 16
fun.diff.c1<-function(bp,bnp,snp){diag(pmax((bp-bnp)^2,snp^2))}
# Zellner, Eq. 15
fun.hot.z2<-function(hot.n,n){if (hot.n < 0.05) (n^2) else 1}
# Zellner-distance, Eq.17
fun.diff.z1<-function(bp,bnp,snp){sqrt(diag(pmax((bp-bnp)^2,snp^2)))}
```

Hotelling's test

Hotelling's test is used in calculating the posterior.

```r
hotelling.test =function(lm_prob, lm_np){
  xbar = lm_prob$coefficients
  mu_0 = lm_np$coefficients
  vcovm = vcov(lm_prob)

  p = length(lm_prob$coefficients)
  n = length(lm_prob$residuals)

  t2 = t(xbar - mu_0)%*%solve(vcovm)%*%(xbar - mu_0)
  f = p*(n-1)/(n-p)
  Fstat = t2/f
  p_val = pf(Fstat,p,n-p,lower.tail = F)
  return(p_val)
}
```

Applying the method

This function takes an input of two lists (such as those resulting from the `gen.sample()` function). The lists represent probability (`sample_prob`) and nonprobability (`sample_np`) data. Each list has two elements: `Y` being the response variable, and `X` - the predictors matrix with a column of ones for intercept.

```r
gen.res = function(sample_prob,sample_np){
  #sample_prob - sample of prob data
  #sample of nprob data
  # conjugate non-inf result
  yp_std=sample_prob$Y
  xp_std=sample_prob$X
  coefsize=dim(xp_std)[2] #number of coefficients
  k=length(yp_std) # length of Prob sample
  result_ni=calc.posterior(mu_0=rep(0,dim(xp_std)[2]), k_0=k, y=yp_std, x=xp_std)
  #ML prob data
  lm_pobj = lm(yp_std ~ xp_std-1)

  # nonprob data
  ynp_std=sample_np$Y
  xnp_std=sample_np$X
  nnp=length(ynp_std) #np sample length

  #ML for nonprob
  lm_npobj = lm(ynp_std ~ xnp_std-1)
  #hotelling test
```

```
hot.n = hotelling.test(lm_pobj,lm_npobj)

#conjugate posterior #k_0fun.hot.c(hot.n,nnp)
result_c = calc.posterior(mu_0 = lm_npobj$coefficients,
                          k_0 = fun.hot.c(hot.n,nnp), y=yp_std,x=xp_std)
#conjugate difference posterior
# fun.diff.c1(lm_pobj$coefficients,
#             lm_npobj$coefficients,
#             summary(lm_npobj)$coefficients[,2])
result_cd = calc.posterior(mu_0 = lm_npobj$coefficients,
                           k_0 = 1/log(nnp),
                           Vin=fun.diff.c1(lm_pobj$coefficients,
                                           lm_npobj$coefficients,
                                           summary(lm_npobj)$coefficients[,2]),
                           y=yp_std,x=xp_std) #
#conjugate Zellner fun.hot.z2(hot.n,nnp)
result_z = calc.posterior(
mu_0 = lm_npobj$coefficients,
k_0 = fun.hot.z2(hot.n,nnp), #changed for the name
y=yp_std,x=xp_std,
Vin = solve(t(xnp_std)%*%xnp_std))

#conjugate Zellner difference fun.diff.z1
result_zd = calc.posterior(
mu_0 = lm_npobj$coefficients,
k_0 = nnp,
y=yp_std,x=xp_std,
Vin = fun.diff.z1(lm_pobj$coefficients,
                  lm_npobj$coefficients,
                  summary(lm_npobj)$coefficients[,2]) %*%
  solve(t(xnp_std)%*%xnp_std) %*%
  fun.diff.z1(lm_pobj$coefficients,
                  lm_npobj$coefficients,
                  summary(lm_npobj)$coefficients[,2]))

# wrapping results
result = rbind(c(k,nnp,
                 result_ni$mu_mean,result_ni$mu_cov,
                 result_c$mu_mean,result_c$mu_cov,
                 result_z$mu_mean,result_z$mu_cov,
                 result_cd$mu_mean,result_cd$mu_cov,
                 result_zd$mu_mean,result_zd$mu_cov,
                 lm_pobj$coefficients,summary(lm_pobj)$coefficients[,2],
                 lm_npobj$coefficients,summary(lm_npobj)$coefficients[,2],hot.n)
          )

#saving as dataframe
par.names = paste0("Beta",c(1:coefsize))
columns.name=c("P_ss","NP_ss",
               #non-inf posterior estimates (mu=mean and se=standard dev)
               paste0("NI.mu.",par.names),paste0("NI.se.",par.names),
               #conjugate
               paste0("C.mu.",par.names),paste0("C.se.",par.names),
                 #Zellner
                 paste0("Z.mu.",par.names),paste0("Z.se.",par.names),
                 #conjugate-difference
                 paste0("CD.mu.",par.names),paste0("CD.se.",par.names),
                 #Zellner difference
                 paste0("ZD.mu.",par.names),paste0("ZD.se.",par.names),
                 #ML for probability data with standard errors
                 paste0("MLP.mu.",par.names),paste0("MLP.se.",par.names),
                 #ML for non-probability data with standard errors
                 paste0("MLNP.mu.",par.names),paste0("MLNP.se.",par.names),"Hotelling.p")
  result=as.data.frame(result)
  colnames(result) = columns.name

  return(result)
}
```

Toy Example

Generating data

Probability sample with parameters c(1, 0.5, .1) and size $n = 50$

```
sample_prob=gen.sample(beta=c(1,0.5,.1), #true coefficients
                       n=50, # sample size
                       m_x=c(0,5), #covariate means
                       sd_response=1, #sd response variable
                       sd_cov1=1, sd_cov2=1, # sd covariates
                       corr=0.1, #correcation of covariates
                       bias=c(1)) #bias= 1 = unbiased
```

Probability sample with size $n = 1000$ and bias in the third coefficient of +25%, i.e. the data are generated with parameters c(1, 0.5, .125)

```
sample_np=gen.sample(beta=c(1,0.5,.1),
                     n=1000, #sample size
                     m_x=c(0,5), #covariate means
                     sd_response=1, ##sd response variable
                     sd_cov1=1, sd_cov2=1,# sd covariates
                     corr=0.1,#correcation of covariates
                     bias=c(1.25)) #bias= 1 = unbiased = here, 25% bias iin third coefficient
```

Producing coefficients

```
simA=gen.res(sample_prob,sample_np)
```

Printing results

```
simA %>% pivot_longer(cols = NI.mu.Beta1:MLNP.se.Beta3, names_to = c("method", "moment",
    "parameter"), names_sep = "\\.", values_to = "value") %>% pivot_wider(names_from = moment,
    values_from = value) %>% View
```

Saving results in a pasteable table

```
simA %>% pivot_longer(cols = NI.mu.Beta1:MLNP.se.Beta3, names_to = c("method", "moment",
    "parameter"), names_sep = "\\.", values_to = "value") %>% pivot_wider(names_from = moment,
    values_from = value) %>% write.table(file = "clipboard")
```