Exemplary R code for EVSI computation algorithms for the manuscript entitled “Expected value of sample information calculations for external validation of clinical prediction models”

[author list embargoed]

2024-01-05

Background

This document is supplementary material for the manuscript entitled “Expected value of sample information calculations for external validation of clinical prediction models”. The purpose of this document is to demonstrates how the proposed algorithms for EVSI computations can be programmed in R.

Please note that the implementations here are for helping the reader better understand the computation algorithms. The implementations are not optimized for speed and computational efficiency. For such implementations please use the open-source evsiexval package https://github.com/resplab/EVSIExVal. Correspondingly, simulations are run with lower numbers than in the paper.

*The code chunks below with default values should take in total 1-10 minutes to run on an average PC.

Data wrangling and general setup

We use data from the GUSTO-I trial. We fit a logistic regression model for predicting 30-day mortality in the non-US sample of GUSTO-I trial. We are interested in validating this model in the US sub-sample. We use a random subset of n=500 from the US sub-sample as the source of our ‘current’ information about the model’s NB, and calculate EVSIs for future validation studies based on these data.

```r
library(predtools) #Contains the GUSTO data. Please install from CRAN or Github
set.seed(123)

data(gusto)
gusto$kill <- (as.numeric(gusto$Killip)>1)*1
gusto$Y <- gusto$day30
data_us <- gusto[gusto$regl %in% c(1, 7, 9, 10, 11, 12, 14, 15),]
data_other <- gusto[!gusto$regl %in% c(1, 7, 9, 10, 11, 12, 14, 15),]

dev_data <- data_other

#This is the risk prediction model
model <- glm(Y~age+miloc+pmi+kill+pmin(sysbp,100)+pulse, data=dev_data,
             family=binomial(link="logit"))

val_data <- data_us[sample(1:(dim(data_us)[1]),500,F),]

#pi is the predicted risk
pi <- predict(model, type="response", newdata=val_data)
val_data$pi <- pi
```

Setting up the overall parameters

We work with z=0.02 threshold, and consider future sample sizes in the 500 - 4000 range.

```
z <- 0.02
future_sample_sizes <- c(500, 1000, 2000, 4000)
n <- nrow(val_data)
```

Bootstrap-based algorithm

For binary outcomes in the absence of censoring and predictors with missing values, this algorithm will converge to the beta-binomial algorithm (which is much faster). The main utility of the algorithm would be in dealing with other types of outcomes or when there is non-ignoble amount of missingness in the data.

Note that in this two-level resampling algorithm, we implement bootstrapping via assigning weights to the observations, instead of creating resamples datasets (e.g., via the sample() function). The latter can involve memory allocation which will slow down the process. Instead, both levels of resampling are represented by assigning weights to each observation.

```r
n_sim <- 10^5

NB0 <- NB1 <- NB2 <- NBbest <-0
NBw <- rep(0, length(future_sample_sizes)) #Max NB after observing future data

#Main simulation loop
for(i in 1:n_sim)
{
  #Bayesian bootstrapping involves sampling from Dirichlet(1,1,...,1).
  #Here we generate weights W by normalizing Gamma random variables.
  W <- rgamma(n, 1, 1)
  W <- W/sum(W)

  #(prev, se, sp) from Bayesian-bootstrapped data taken as draws from their posterior dist.
  prev <- sum(val_data$Y*W)/sum(W)
  se <- sum(val_data$Y*(val_data$pi>=z)*W)/sum(val_data$Y*W)
  sp <- sum((1-val_data$Y)*(val_data$pi<z)*W)/sum((1-val_data$Y)*W)

  #These are draws from 'true' NBs
  NB1t <- prev*se-(1-prev)*(1-sp)*z/(1-z)
  NB2t <- prev-(1-prev)*z/(1-z)
  NBbest <- NBbest + max(0, NB1t, NB2t) #For EVPI computation
  #Need to collect the sum to calculate ENB_current_info
  #As stated in the paper, this converges to NB calculated directly from the sample
  #However, embedding calculations inside the loop prevents getting negative VoI quantities
  #due to MC error
  NB1 <- NB1 + NB1t
  NB2 <- NB2 + NB2t


  for(j in 1:length(future_sample_sizes)) #Loop over requested future sample sizes
  {
    n_star <- future_sample_sizes[j]
    #Second-level resampling creates D*
    W_star <- rmultinom(1, n_star, W)
    #Adding +1 to all weights represents adding the current sample to the future sample
    W_pooled <- W_star+1

    #Imputing missing predictor values should be implemented at this point (not relevant for GUSTO data

    prev_pooled <- sum(val_data$Y*W_pooled)/sum(W_pooled)
    se_pooled <- sum(val_data$Y*(val_data$pi>=z)*W_pooled)/sum(val_data$Y*W_pooled)
    sp_pooled <- sum((1-val_data$Y)*(val_data$pi<z)*W_pooled)/sum((1-val_data$Y)*W_pooled)
    #NB0 <- 0
    NB1_pooled <- prev_pooled*se_pooled-(1-prev_pooled)*(1-sp_pooled)*z/(1-z)
    NB2_pooled <- prev_pooled-(1-prev_pooled)*z/(1-z)

    NBw[j] <- NBw[j]+max(c(0,NB1_pooled,NB2_pooled))
    #Note: a variance reduction technique is to use
    #NBw[j] <- NBw[j]+c(0,NB1t,NB2t)[which.max(c(0,NB1_pooled,NB2_pooled))]
    #This is because th
  }
}

ENB_current_info <- max(0, NB1/n_sim, NB2/n_sim)

EVPI <- NBbest/n_sim - ENB_current_info
EVSI <- NBw/n_sim - ENB_current_info
```

Showing the results

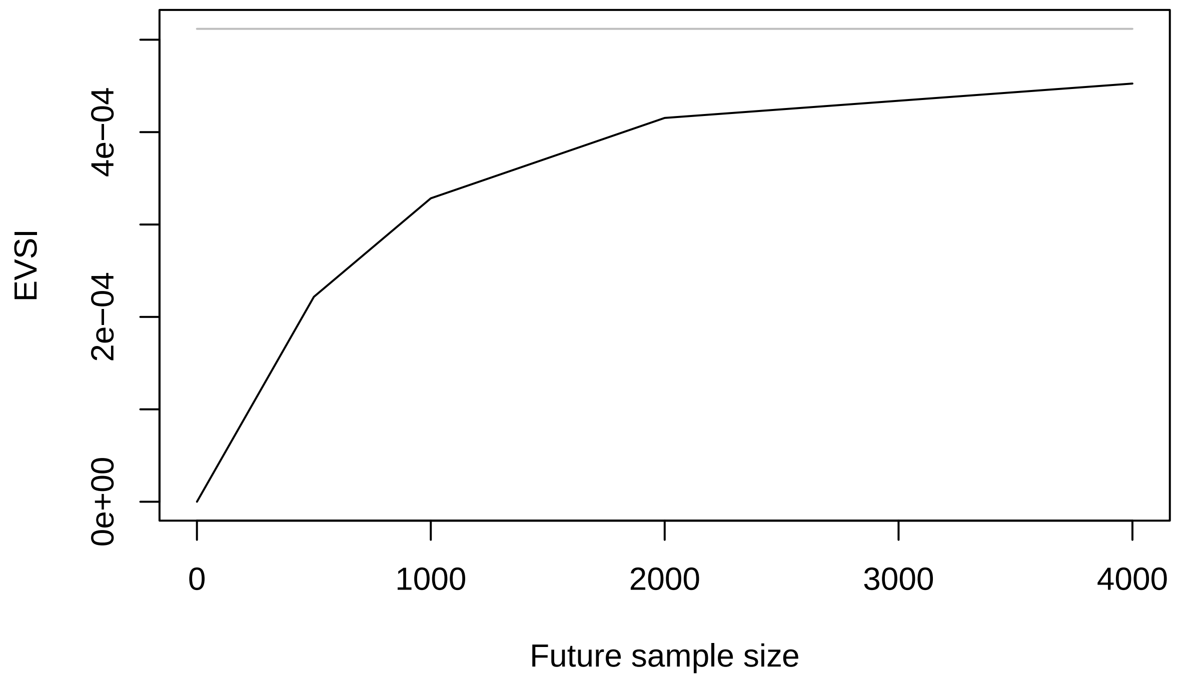
```
print(EVPI)
```

```
## [1] 0.0005117537
```

```
print(EVSI)
```

```
## [1] 0.0002217293 0.0003283742 0.0004153565 0.0004525525
```

```
plot(c(0,future_sample_sizes), c(0,EVSI), type='l', ylim=c(0,EVPI),
      xlab="Future sample size", ylab="EVSI")
lines(c(0,max(future_sample_sizes)),c(EVPI,EVPI),type='l',col='gray')
```


EVSI
0e+00
2e−04
4e−04
0
1000
2000
3000
4000
Future sample size

Beta-binomial method

For binary outcomes, the bootstrapped-based algorithm can be done in aggregate due to the beta-binomial conjugacy.

Instead of raw data, this algorithm works with aggregate frequencies. As stated in the paper, the sample can be summarized by its sample size as well as outcome prevalence, sensitivity, and specificty, which all have beta distirbutions.

The implementation of this code in the evsiexval package is vectorized. Unvectorized R code is provided here because it is easier to process.

```
n_sim <- 100000

n <- nrow(val_data)
n_D <- sum(val_data$Y)
n_TP <- sum(val_data$Y*(val_data$pi>=z))
n_FN <- n_D-n_TP
n_TN <- sum((1-val_data$Y)*(val_data$pi<z))
n_FP <- n-n_D-n_TN

evidence <- list(prev=c(n_D, n-n_D),
              se=c(n_TP, n_FN),
              sp=c(n_TN, n_FP))

print(paste("alpha and beta parameters of prev, se, and sp (in order):", paste(evidence, collapse=" | ")
```

```
## [1] "alpha and beta parameters of prev, se, and sp (in order): c(43, 457) | c(41, 2) | c(147, 310)"
```

NOTE: To keep the results consistent with the bootstrap method, we are assigning Beta(0.0) to all three parameters (which is also the implied prior in the Bayesian bootstrap). Note that the results in the paper are based on Beta(1,1) priors.

```
#Moving from evidence d to posterior P(theta|d), assuming prior Beta(0,0) on all parameters.
#This line copies evidence to posterior because of Beta(0,0) assumptions
posterior <- list(prev=evidence$prev+c(0,0), se=evidence$se+c(0,0), sp=evidence$sp+c(0,0))

prev <- posterior$prev[1]/sum(posterior$prev)
se <- posterior$se[1]/sum(posterior$se)
sp <- posterior$sp[1]/sum(posterior$sp)

NB0 <- NB1 <- NB2 <- NBbest <-0
NBw <- rep(0, length(future_sample_sizes))

for(i in 1:n_sim)
{
  prev <- rbeta(1, posterior$prev[1], posterior$prev[2])
  se <- rbeta(1, posterior$se[1], posterior$se[2])
  sp <- rbeta(1, posterior$sp[1], posterior$sp[2])

  NB1t <- prev*se-(1-prev)*(1-sp)*z/(1-z)
  NB2t <- prev-(1-prev)*z/(1-z)
  NBbest <- NBbest + max(0, NB1t, NB2t)
  NB1 <- NB1 + NB1t
  NB2 <- NB2 + NB2t

  for(j in 1:length(future_sample_sizes))
```

```r
  {
    #Generating D* which can be summarized in terms of true/false positive/negative frequencies
    n_star <- future_sample_sizes[j]
    n_D_star <- rbinom(1, size=n_star, prob=prev)
    n_TP_star <- rbinom(1, size=n_D_star, prob=se)
    n_TN_star <- rbinom(1, size=n_star-n_D_star, prob=sp)

    #pooling future and current samples
    prev_pooled <- (n_D+n_D_star)/(n+n_star)
    se_pooled <- (n_TP+n_TP_star)/(n_D+n_D_star)
    sp_pooed <- (n_TN+n_TN_star)/(n-n_D+n_star-n_D_star)

    NB1_pooled <- prev_pooled*se_pooled-(1-prev_pooled)*(1-sp_pooed)*z/(1-z)
    NB2_pooled <- prev_pooled-(1-prev_pooled)*z/(1-z)

    NBw[j] <- NBw[j]+c(0,NB1t,NB2t)[which.max(c(0,NB1_pooled,NB2_pooled))]
    #Note: a variance reduction technique is to use
    #NBw[j] <- NBw[j]+c(0,NB1t,NB2t)[which.max(c(0,NB1_pooled,NB2_pooled))]
  }
}

ENB_current_info <- max(0, NB1/n_sim, NB2/n_sim)

EVPI <- NBbest/n_sim - ENB_current_info
EVSI <- NBw/n_sim - ENB_current_info
```

Showing the results

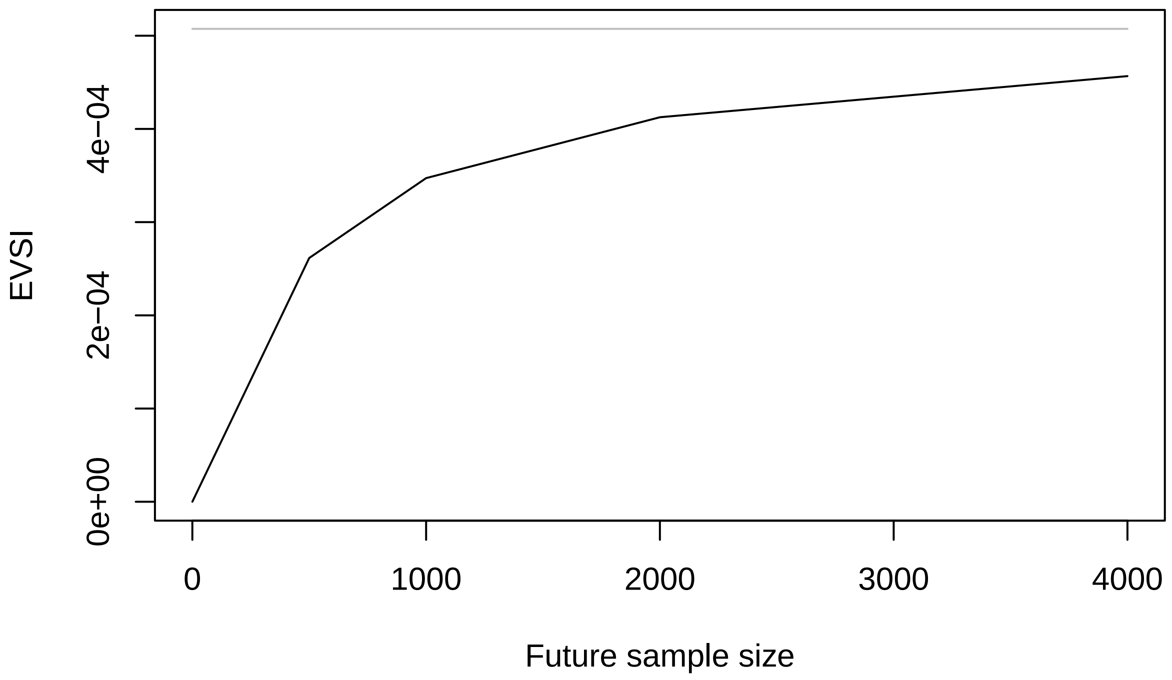
```r
print(EVPI)
```

```
## [1] 0.0005073819
```

```r
print(EVSI)
```

```
## [1] 0.0002615217 0.0003472299 0.0004125427 0.0004566638
```

```r
plot(c(0,future_sample_sizes), c(0,EVSI), type='l', ylim=c(0,EVPI),
      xlab="Future sample size", ylab="EVSI")
lines(c(0,max(future_sample_sizes)),c(EVPI,EVPI),type='l',col='gray')
```


EVSI
0e+00
2e−04
4e−04
0
1000
2000
3000
4000
Future sample size

Generic, sample-based method

This algorithm is a general one that does not require the current information P(theta|d) to be in an expressible mathematical form. Instead, it requires a sample from the (joint) posterior distribution of (prev, se, sp). This is particularly relevant for situations where we are using MCMC methods to generate posterior samples from a complex model (e.g., a random-effects model for joint inference on sensitivity and specificity of the model at the threshold of interest).

This algorithm has a complexity of $O(N_{out}.N_{in}.M)$ with terms representing, respectively, the number of outer and inner simulations and the size of the sample. This can quickly become overwhelming for R. The evsiexval package implements this algorithm in C++ (which is used for the results reported in the paper).

The R code below is inevitably vectorized to avoid very long times, and the number of inner simulations is kept at 1000 and the sample at 100. The sample of 100 rows is not generally sufficient and below results are only for demonstrating purposes.

Instead of an outer simulation that samples from each observation, we are looping over the 100 observations. This removes the unnecessary Monte Carlo error for the outer Monte Carlo simulation. However, this will not be practical if the sample from the posterior distribution is large.

We start by generating M samples from {prev, se, sp}

```r
n_sim <- 1000
M <- 100
samples <- cbind(prev=rbeta(M, evidence$prev[1],evidence$prev[2]),
                se=rbeta(M, evidence$se[1],evidence$se[2]),
                sp=rbeta(M, evidence$sp[1],evidence$sp[2])
                )
```

```r
#Vectorized calculation of NBs
NBt <- cbind(0,
        samples[,1]*samples[,2]-(1-samples[,1])*(1-samples[,3])*z/(1-z),
        samples[,1]-(1-samples[,1])*z/(1-z)
)

EVPI <- mean(apply(NBt,1,max)) - max(colMeans(NBt))

#Replicating the sample n_sim times such that we can vectorize the calculations
#WARNING: This part is RAM-intensive as it creates matrices with M*n_sim rows
#We are looping over the M samples instead of randomly drawing from them
#This stratified sampling removes Monte Carlo error due to the outer simulatio loop
#If M is large and this will not be practical,
#in which case modify this part to sample from a smaller subset of the sample
S <- do.call(rbind, replicate(n_sim, samples, simplify=FALSE))

EVSI <- double(length(future_sample_sizes))
for(j in 1:length(future_sample_sizes))
{
  cat('.') #Give some sense of progress given it might take some time to complete.

  #Data for the future study for every row of S
  n_star <- future_sample_sizes[j]
  n_D_star <- rbinom(nrow(S), size=n_star, prob=S[,1])
  n_TP_star <- rbinom(nrow(S), size=n_D_star, prob=S[,2])
  n_TN_star <- rbinom(nrow(S), size=n_star-n_D_star, prob=S[,3])

  #Creating the weight vector outside the function to minimize memory reallocation
  w <- double(nrow(samples))
  NB_pooled <- matrix(double(1), nrow=nrow(samples),ncol=3)

  #Takes one realization of future study and updates the evidence and returns the highest NB
  find_winner <- function(D_TP_TN)
  {
    #This is the likelihood
    w <<- dbinom(D_TP_TN[1], n_star, samples[,1])*
      dbinom(D_TP_TN[2], D_TP_TN[1], samples[,2])*
      dbinom(D_TP_TN[3], n_star-D_TP_TN[1], samples[,3])

    max(0,
      sum(w*(samples[,1]*samples[,2]-(1-samples[,1])*(1-samples[,3])*z/(1-z)))/sum(w),
      sum(w*(samples[,1]-(1-samples[,1])*z/(1-z))/sum(w))
    )
  }

  NBw <- apply(cbind(n_D_star, n_TP_star, n_TN_star), 1, find_winner)

  EVSI[j] <-  mean(NBw) - max(colMeans(NBt))
}
```

```
## ....
```

Plotting the results

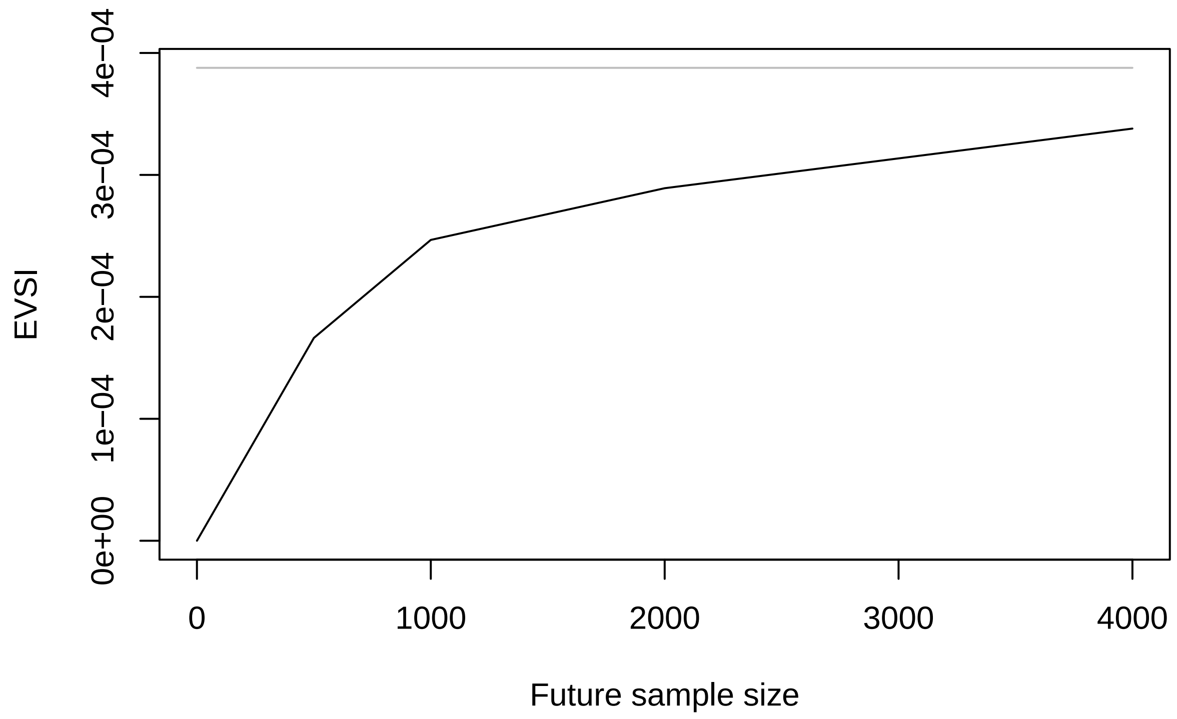
```
print(EVPI)
```

```
## [1] 0.0003878008
```

```
print(EVSI)
```

```
## [1] 0.0001663050 0.0002467015 0.0002890922 0.0003379857
```

```
plot(c(0,future_sample_sizes), c(0,EVSI), type='l', ylim=c(0,EVPI),
      xlab="Future sample size", ylab="EVSI")
lines(c(0,max(future_sample_sizes)),c(EVPI,EVPI),type='l',col='gray')
```


EVSI
0e+00
1e−04
2e−04
3e−04
4e−04
0
1000
2000
3000
4000
Future sample size