C-Index Simulation Study

Your Name

2024-07-01

Table of contents

## Introduction

This document contains a simulation study comparing different methods for estimating the C-index.

## Load Required Packages

# List of required packages  
required\_packages <- c("rms", "dplyr", "winprob", "sandwich", "lmtest", "broom", "ggplot2", "parallel", "pbapply", "viridis")  
  
# Function to check and install missing packages  
check\_and\_install\_packages <- function(packages) {  
 install\_if\_missing <- function(pkg) {  
 if (!require(pkg, character.only = TRUE)) {  
 if (pkg == "winprob") {  
 if (!requireNamespace("devtools", quietly = TRUE)) install.packages("devtools")  
 devtools::install\_github("proshano/winprob")  
 } else {  
 install.packages(pkg)  
 }  
 library(pkg, character.only = TRUE)  
 }  
 }  
 invisible(lapply(packages, install\_if\_missing))  
}  
  
# Run the function to check and install missing packages  
check\_and\_install\_packages(required\_packages)  
  
# Load necessary libraries  
library(rms)  
library(dplyr)  
library(winprob)  
library(sandwich)  
library(lmtest)  
library(broom)  
library(ggplot2)  
library(parallel)  
library(pbapply)  
library(viridis)

## Set Random Seed

# Set the random seed for reproducibility  
set.seed(1)

## Define Simulation Function

# Define the simulation function  
run\_simulation <- function(total\_n, ratio, var2, var\_ratio, num\_runs) {  
 results <- data.frame()  
 valid\_runs <- 0  
   
 for (i in 1:num\_runs) {  
 tryCatch({  
 # Calculate sample sizes and variances  
 n1 <- round(total\_n \* ratio / (1 + ratio))  
 n2 <- total\_n - n1  
 var1 <- var2 \* var\_ratio  
   
 # Generate data  
 y1 <- rnorm(n1, 0, sqrt(var1))  
 y2 <- rnorm(n2, 1, sqrt(var2))  
 group <- c(rep(0, n1), rep(1, n2))  
 y <- c(y1, y2)  
 sim\_data <- data.frame(y = y, group = factor(group))  
   
 # Calculate sample concordance  
 conc <- (mean(rank(y)[group == 1]) - (n2 + 1) / 2) / n1  
   
 # Calculate C-index approximation  
 f <- orm(y ~ group, data = sim\_data, x = TRUE, y = TRUE)  
 or <- exp(coef(f)['group=1'])  
 capprox <- or^0.66 / (1 + or^0.66)  
   
 # Calculate CI for C-index approximation  
 se\_log\_or <- sqrt(diag(vcov(f)))['group=1']  
 dcapprox\_dor <- 0.66 \* or^(-0.34) / (1 + or^0.66)^2  
 se\_capprox <- abs(dcapprox\_dor) \* or \* se\_log\_or  
 ci\_lower\_approx <- capprox - 1.96 \* se\_capprox  
 ci\_upper\_approx <- capprox + 1.96 \* se\_capprox  
   
 # Calculate WinP  
 sim\_data$group\_swapped <- ifelse(sim\_data$group == 0, 1, 0)  
 wp\_result <- calculate\_winP(data = sim\_data, group\_var = "group\_swapped", post\_var = "y")  
 winP <- wp\_result$WinP  
 ci\_lower\_winP <- wp\_result$WinP\_l  
 ci\_upper\_winP <- wp\_result$WinP\_u  
   
 # Normal theory-based CI for concordance  
 var\_conc <- (conc \* (1 - conc) + (n2 - 1) \* (0.5 - conc)^2 + (n1 - 1) \* (0.5 - (1 - conc))^2) / (n1 \* n2)  
 se\_conc <- sqrt(var\_conc)  
 ci\_lower\_conc\_normal <- conc - 1.96 \* se\_conc  
 ci\_upper\_conc\_normal <- conc + 1.96 \* se\_conc  
   
 # Bootstrap CI for concordance  
 boot\_conc <- replicate(1000, {  
 boot\_sample <- sample(length(y), replace = TRUE)  
 boot\_y <- y[boot\_sample]  
 boot\_group <- group[boot\_sample]  
 (mean(rank(boot\_y)[boot\_group == 1]) - (sum(boot\_group == 1) + 1) / 2) / sum(boot\_group == 0)  
 })  
 ci\_conc\_boot <- quantile(boot\_conc, c(0.025, 0.975))  
   
 # Calculate true concordance  
 delta <- 1 / sqrt((var1 + var2) / 2)  
 true\_conc <- pnorm(delta / sqrt(2))  
   
 # Calculate biases  
 bias\_approx <- (capprox - true\_conc) / true\_conc \* 100  
 bias\_winP <- (winP - true\_conc) / true\_conc \* 100  
 bias\_conc <- (conc - true\_conc) / true\_conc \* 100  
   
 # Calculate coverage  
 coverage\_approx <- (true\_conc >= ci\_lower\_approx) && (true\_conc <= ci\_upper\_approx)  
 coverage\_winP <- (true\_conc >= ci\_lower\_winP) && (true\_conc <= ci\_upper\_winP)  
 coverage\_conc\_normal <- (true\_conc >= ci\_lower\_conc\_normal) && (true\_conc <= ci\_upper\_conc\_normal)  
 coverage\_conc\_boot <- (true\_conc >= ci\_conc\_boot[1]) && (true\_conc <= ci\_conc\_boot[2])  
   
 # Store results  
 results <- rbind(results, data.frame(  
 Run = i,  
 TotalSampleSize = total\_n,  
 Ratio = ratio,  
 VarRatio = var\_ratio,  
 TrueConc = true\_conc,  
 BiasApprox = bias\_approx,  
 BiasWinP = bias\_winP,  
 BiasConc = bias\_conc,  
 CoverageApprox = coverage\_approx,  
 CoverageWinP = coverage\_winP,  
 CoverageConcNormal = coverage\_conc\_normal,  
 CoverageConcBoot = coverage\_conc\_boot  
 ))  
   
 valid\_runs <- valid\_runs + 1  
 }, error = function(e) {  
 cat("Error in run", i, ":", conditionMessage(e), "\n")  
 })  
 }  
   
 return(list(results = results, valid\_runs = valid\_runs))  
}

## Set Simulation Parameters

# Define parameters  
num\_runs <- 1825 # Number of simulation runs for each sample size  
ratios <- c(.5, 1, 2,5) # Ratios of n1 to n2  
total\_sample\_sizes <- c(100, 500, 1000, 2000) # Sequence of total sample sizes  
var2 <- 1 # Fixed variance for group 2  
var\_ratios <- c(0.5, 1, 2,5) # Variance ratios for group 1 compared to group 2 (var1 = var2 \* ratio)  
  
# Create a list of all parameter combinations  
param\_combinations <- expand.grid(  
 total\_n = total\_sample\_sizes,  
 ratio = ratios,  
 var\_ratio = var\_ratios  
)

## Run Simulations

# Set up parallel processing  
num\_cores <- 9  
cl <- makeCluster(num\_cores)  
clusterExport(cl, c("run\_simulation", "num\_runs", "var2", "param\_combinations", "calculate\_winP"))  
clusterEvalQ(cl, {  
 library(rms)  
 library(dplyr)  
 library(winprob)  
 library(sandwich)  
 library(lmtest)  
 library(broom)  
 library(pbapply)  
})  
clusterSetRNGStream(cl, 12345)  
  
# Run simulations in parallel  
all\_results <- pblapply(1:nrow(param\_combinations), function(i) {  
 total\_n <- param\_combinations$total\_n[i]  
 ratio <- param\_combinations$ratio[i]  
 var\_ratio <- param\_combinations$var\_ratio[i]  
   
 cat("\nStarting simulation with parameters:\n")  
 cat("Total N:", total\_n, "\n")  
 cat("Ratio:", ratio, "\n")  
 cat("Var2:", var2, "\n")  
 cat("Var Ratio:", var\_ratio, "\n")  
 cat("Number of runs:", num\_runs, "\n\n")  
   
 sim\_result <- run\_simulation(total\_n, ratio, var2, var\_ratio, num\_runs)  
   
 # Calculate summary statistics  
 summary\_stats <- sim\_result$results %>%  
 summarise(across(c(TrueConc, BiasApprox, BiasWinP, BiasConc,   
 CoverageApprox, CoverageWinP, CoverageConcNormal, CoverageConcBoot),  
 mean))  
   
 summary\_stats$TotalSampleSize <- total\_n  
 summary\_stats$Ratio <- ratio  
 summary\_stats$VarRatio <- var\_ratio  
 summary\_stats$ValidRuns <- sim\_result$valid\_runs  
   
 return(summary\_stats)  
}, cl = cl)  
  
# Stop the cluster  
stopCluster(cl)  
  
# Combine results  
all\_results <- do.call(rbind, all\_results)

## Save Results

# Save results  
write.csv(all\_results, "simulation\_summary\_results.csv", row.names = FALSE)  
  
cat("\nResults saved to 'simulation\_summary\_results.csv'\n")

## Visualization

# Convert Ratio and VarRatio to factors for better plotting  
all\_results$Ratio <- factor(all\_results$Ratio, levels = ratios)  
all\_results$VarRatio <- factor(all\_results$VarRatio, levels = var\_ratios)  
  
# Create custom labeller  
custom\_labeller <- labeller(  
 Ratio = function(value) {  
 paste("Sample Size Ratio:", value)  
 },  
 VarRatio = function(value) {  
 paste("Variance Ratio:", value)  
 }  
)  
  
# Define colorblind-friendly palette using viridis  
colorblind\_palette <- viridis(4)  
  
# Create plot for bias  
plot\_bias <- ggplot(all\_results, aes(x = TotalSampleSize)) +  
 geom\_line(aes(y = BiasApprox, color = "PO Model", linetype = "PO Model"), size = 1) +  
 geom\_line(aes(y = BiasWinP, color = "WinP", linetype = "WinP"), size = 1) +  
 geom\_line(aes(y = BiasConc, color = "Calculated", linetype = "Calculated"), size = 1) +  
 geom\_hline(yintercept = 0, linetype = "dashed", color = "gray") +  
 facet\_grid(Ratio ~ VarRatio, labeller = custom\_labeller) +  
 scale\_color\_manual(values = colorblind\_palette) +  
 scale\_linetype\_manual(values = c("solid", "dashed", "dotted", "dotdash")) +  
 labs(title = "Bias Comparison",   
 x = "Total Sample Size",   
 y = "Bias (%)",   
 color = "Method",   
 linetype = "Method") +  
 theme\_minimal() +  
 theme(legend.position = "bottom")  
  
# Create plot for coverage  
plot\_coverage <- ggplot(all\_results, aes(x = TotalSampleSize)) +  
 geom\_line(aes(y = CoverageApprox, color = "PO Model", linetype = "PO Model"), size = 1) +  
 geom\_line(aes(y = CoverageWinP, color = "WinP", linetype = "WinP"), size = 1) +  
 geom\_line(aes(y = CoverageConcNormal, color = "Normal (Calculated)", linetype = "Normal (Calculated)"), size = 1) +  
 geom\_line(aes(y = CoverageConcBoot, color = "Bootstrap (Calculated)", linetype = "Bootstrap (Calculated)"), size = 1) +  
 geom\_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  
 facet\_grid(Ratio ~ VarRatio, labeller = custom\_labeller) +  
 scale\_color\_manual(values = colorblind\_palette) +  
 scale\_linetype\_manual(values = c("solid", "dashed", "dotted", "dotdash")) +  
 labs(title = "Coverage Probability Comparison",  
 x = "Total Sample Size",  
 y = "Coverage Probability",  
 color = "Method",  
 linetype = "Method") +  
 theme\_minimal() +  
 theme(legend.position = "bottom")  
  
# Display the plots  
print(plot\_bias)  
print(plot\_coverage)  
  
# Save the plots  
ggsave("c\_index\_bias\_comparison\_plot.jpg", plot\_bias, width = 12, height = 10, dpi = 300)  
ggsave("c\_index\_coverage\_comparison\_plot.jpg", plot\_coverage, width = 12, height = 10, dpi = 300)

## Final Summary Statistics

# Print final summary statistics  
summary\_stats <- all\_results %>%  
 group\_by(TotalSampleSize, Ratio, VarRatio) %>%  
 summarise(across(c(TrueConc,   
 BiasApprox,   
 BiasWinP,   
 BiasConc,  
 CoverageApprox,   
 CoverageWinP,   
 CoverageConcNormal,   
 CoverageConcBoot),   
 mean, .names = "mean\_{.col}"),  
 ValidRuns = first(ValidRuns),  
 .groups = 'drop') %>%  
 rename(  
 `Bias PO Model` = mean\_BiasApprox,  
`Bias WinP` = mean\_BiasWinP,  
 `Bias Normal` = mean\_BiasConc,  
 `Coverage PO Model` = mean\_CoverageApprox,  
 `Coverage WinP` = mean\_CoverageWinP,  
 `Coverage Normal` = mean\_CoverageConcNormal,  
 `Coverage Bootstrap` = mean\_CoverageConcBoot  
 ) %>%  
 arrange(TotalSampleSize, Ratio, VarRatio)  
  
print(summary\_stats)  
  
# Save the summary statistics  
write.csv(summary\_stats, "final\_summary\_statistics.csv", row.names = FALSE)

This completes the Quarto document with all the original code and added Quarto-specific elements. The document now includes:

1. YAML header with Quarto-specific options
2. Code chunks with Quarto options (e.g., #| eval: false for chunks that shouldn’t be evaluated when rendering)
3. Markdown text to provide context and structure to the document

To use this Quarto document:

1. Save it as a .qmd file (e.g., “c\_index\_simulation.qmd”)
2. Open it in RStudio or your preferred Quarto-compatible IDE
3. Install the necessary packages if you haven’t already
4. Render the document using the “Render” button in RStudio or by running quarto render c\_index\_simulation.qmd in the terminal

Note that I’ve set eval: false for the chunks that run the simulations and create plots. This is because these operations are time-consuming and may not be necessary every time you render the document. You can change these to eval: true when you want to run the full simulation and generate new results.