Evaluating the performance of the proportional odds model in estimating Zou's Win Probability.

Pavel Roshanov & GY Zou

2024-07-02

Table of contents

Introduction

This document contains a simulation study comparing different methods for estimating the Win Probability.

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
```

```r
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 Simulation Parameters

Estimates for each parameter combination are calculated in 1825 samples.

```r
# 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
```

```r
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
)
```

Set Random Seed

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

Define Simulation Function

```r
# 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
```

```r
    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))
}
```

Run Simulations

Simulations take advantage of parallel processing.

```
# Set up parallel processing
num_cores <- detectCores() - 1
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)
```

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.