PHP 2550: Worksheet 11

Due: November 22nd at 11:59pm

Paper Summaries

1. The paper "Using Simulation Studies to Evaluate Statistical Methods" summarizes the ADEMP framework for designing simulation studies with Monte Carlo simulation.

    - Summarize the ADEMP setup for the worked example in Section 7. Be sure to clearly specify each piece.

Aims

The simulation study aimed to evaluate the impacts of model misspecification on estimating the treatment effect (log-hazard ratio, (θ)) in a randomized trial with survival outcomes. Specifically, it explored:

1. The effect of misspecifying the baseline hazard function.
2. The implications of fitting an overly complex model when simpler ones suffice.
3. The benefits of using a semi-parametric model to avoid the above issues.

Data-Generating Mechanisms

The study used two data-generating mechanisms:

1. ($\lambda = 0.1$, $\gamma = 1$): Represents both exponential and Weibull models.
2. ($\lambda = 0.1$, $\gamma = 1.5$): Represents a Weibull model but not an exponential model.

Survival times were simulated for (n = 500) patients using proportional hazards models, with treatment assignment ((X_i)) determined via randomization ((Bernoulli(0.5))). The true log-hazard ratio ((θ)) was set to -0.5, corresponding to a hazard ratio of 0.607.

Estimands

The primary estimand was the log-hazard ratio ((θ)) for treatment versus control.

Methods

Three methods were compared:

1. Exponential model.
2. Weibull model.
3. Cox proportional hazards model (semi-parametric).

Performance Measures

Bias, empirical standard error, coverage, and relative precision were assessed.

- How were the results analyzed and presented? (1 paragraph)

  The results were presented both in tabular and graphical formats. Tables displayed performance measures including bias and coverage for the two data-generating mechanisms and methods, with Monte Carlo standard errors provided in parentheses. Lollipop plots visually compared the methods, stacking performance measures vertically and using confidence intervals for clarity. The findings highlighted that model misspecification could result in bias but that coverage remained adequate due to overestimation of standard errors.

- How were the number of simulation repetitions determined? (2-3 sentences)

  The number of repetitions was chosen to ensure acceptable Monte Carlo standard errors for key performance measures. Specifically, 1600 repetitions were determined to produce sufficiently small errors for reliable conclusions.

2. The paper "Best Subset, Forward Stepwise or Lasso" uses a simulation study to compare between common forms for sparse variable estimation.

   - Define the different models the authors are comparing. (1 paragraph each)

Best subset selection

This model identifies the subset of predictors that minimizes the residual sum of squares for a fixed subset size k. This method is the most computationally intensive among the methods studied due to the combinatorial optimization problem it solves. The paper uses the Mixed Integer Optimization framework, which dramatically improves the computational feasibility for larger p, allowing the authors to include this method in the simulation study.

Forward stepwise selection

This is an iterative method that begins with no predictors in the model and sequentially adds predictors based on the highest improvement in model fit. This method balances computational efficiency and flexibility, often producing models similar to Best Subset Selection while being substantially faster.

Lasso

The Lasso applies an L1 penalty to the regression coefficients, encouraging sparsity. The Lasso's computational advantage lies in its convex formulation, making it faster to compute compared to Best Subset Selection. However, Lasso introduces shrinkage to the selected coefficients, potentially introducing bias in the estimates.

Relaxed Lasso

The Relaxed Lasso modifies the Lasso's estimates by reducing the shrinkage of coefficients after the initial selection. By tuning an additional parameter gamma, the Relaxed Lasso creates a balance between the sparse selection of Lasso and the unbiased estimation of the least squares method

- Use the ADEMP framework to define the simulation study the authors used to compare these methods and summarize the results.

Aim

The simulation study aims to compare the prediction accuracy and variable selection performance of Best Subset, Forward Stepwise, Lasso, and Relaxed Lasso across various settings of signal-to-noise ratio, sparsity levels, predictor correlation, and the number of predictors.

Data generation

Simulated datasets were generated with n observations and p predictors. The value of n,p, and s are varied.

Estimands

The estimands include prediction accuracy, measured via metrics like Relative Test Error and Proportion of Variance Explained, as well as variable selection accuracy, assessed using F-scores and the number of nonzero coefficients.

Methods

The four methods are compared:Best Subset, Forward Stepwise, Lasso, Relaxed Lassp. Models were tuned using external validation sets to minimize prediction error.

Performance

The study evaluates RTE, PVE, F-scores, and computational cost.

- Give your own evaluation of the simulation study. (1-2 paragraphs)

  The simulation study does a good job of comparing the methods under various scenarios, providing clarity on their strengths and weaknesses. It's well-structured, using a mix of performance metrics to evaluate both predictive accuracy and sparsity. Including settings with different SNRs, correlation levels, and sparsity patterns ensures the results are applicable to a wide range of practical problems. The decision to focus on key metrics like RTE and F-scores is smart because it simplifies the interpretation of results without overcomplicating the analysis.

  That said, there are a few issues. The reliance on external validation for tuning parameters might inherently favor computationally simpler methods like Lasso and Relaxed Lasso, making the results slightly biased against Best Subset. Also, while the paper focuses on these four methods, adding comparisons with other sparse regression techniques like SCAD or Elastic Net might have made the conclusions more robust. Computational cost, especially for Best Subset, is a real problem—it's not just theoretical, but practically unusable in larger settings. Overall, though, the study achieves its aim of highlighting the trade-offs between these canonical approaches, with Relaxed Lasso emerging as the most balanced and effective choice.

3. The paper "Projected Estimates of Opioid Mortality After Community-Level Interventions" provides an example of a different type of simulation study. Give an overview of the type of model used and why it was appropriate for the given research question. (1-2 paragraphs)

Variable Selection Simulation Study

Variable selection using p-values has some limitations. The test statistic for a single β coefficient in multiple linear regression is given below.

$$\frac{\hat{\beta}_j}{\text{se}(\beta_j)} = \hat{\beta}_j \frac{\sqrt{n\widehat{Var}(x_j)}}{\hat{\sigma}\sqrt{\text{VIF}_j}}$$

Looking at the equation, we can observe what makes this test statistic larger (or smaller). For example, larger estimated coefficients have larger test statistics and are more likely to be significant.

The point of this question is to illustrate that reported regression coefficients and associated standard errors can be incorrect when selecting variables by p-values. We will illustrate this concept in a simple case. Consider a simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \text{ where } \epsilon_i \sim N(0, 1)$$

Suppose you have $m = 100$ observations with observed x and y. Your interest centers on estimating β_1. Consider the following estimation procedure. You fit a simple linear regression model and test H0 : $\beta_1 = 0$ vs H1 : $\beta_1 \neq 0$ at level of significance α. If the test fails to reject H0, you drop x from the model and set the estimator to 0. If the test rejects H0, set the estimator to the OLS estimator $\hat{\beta}$. Call this estimator $\hat{\beta}_\alpha$. Evaluate the performance of this estimator using a simulation study. Use the ADEMP framework.

We will follow the ADEMP framework.

Aim:

Illustrate that reported regression coefficients and associated standard errors can be incorrect when selecting variables by p-values.

Data-generating mechanisms:

In this step, we define how the data for the simulation will be generated. We will follow the mentioned simple linear regression model. Notice that epsilon and m are fixed but we would like to vary β_1 and the variance of x which are presented in the formula of deriving p-value. We also fix β_0.

```r
# Function to generate data
generate_data <- function(beta_1, sigma_x) {
  m <- 100  # Fixed sample size
  beta_0 <- 0  # Fixed intercept

  x <- rnorm(m, mean = 0, sd = sqrt(sigma_x))  # Predictor
  epsilon <- rnorm(m, mean = 0, sd = 1)  # Noise
  y <- beta_0 + beta_1 * x + epsilon  # Response
  data.frame(x = x, y = y)
}
```

Estimand:

The estimand in our study is the regression coefficient β_1. We will use OLS to estimate this value, while setting the value to 0 is p-value is above threshold of 0.05.

```r
# Function to estimate beta1 and compute performance metrics
estimate_beta1 <- function(data, beta_true, alpha = 0.05) {
  model <- lm(y ~ x, data = data)
  se_beta_1 <- summary(model)$coefficients["x", "Std. Error"]
  p_value <- summary(model)$coefficients["x", "Pr(>|t|)"]
  if (p_value < alpha) {
    beta_1 <- coef(model)["x"]
  } else {
    beta_1 <- 0
  }
  ci_lower <- beta_1 - qt(1 - alpha / 2, df = nrow(data) - 2) * se_beta_1
  ci_upper <- beta_1 + qt(1 - alpha / 2, df = nrow(data) - 2) * se_beta_1
  coverage <- ifelse(ci_lower <= beta_true & beta_true <= ci_upper, 1, 0)
  list(beta_1 = beta_1, se_beta1 = se_beta_1, p_value = p_value,
       coverage = coverage)
}
```

Method:

We will calculate bias, empirical SE,MSE, and coverage.

```r
# Function to run simulation for a single parameter combination
run_simulation <- function(beta_1, sigma_x, n_sim, alpha = 0.05) {
  results <- replicate(n_sim, {
    # Generate data
    data <- generate_data(beta_1 = beta_1, sigma_x = sigma_x)
    # Estimate beta1 and compute metrics
    estimate_beta1(data, beta_true = beta_1, alpha = alpha)
  }, simplify = FALSE)

  # Combine into a data frame
  results_df <- do.call(rbind, lapply(results, as.data.frame))

  # Calculate performance metrics
  bias <- mean(results_df$beta_1) - beta_1
  empirical_se <- sd(results_df$beta_1)
  mse <- mean((results_df$beta_1 - beta_1)^2)
  coverage <- mean(results_df$coverage)

  # Return as a single row data frame
  data.frame(beta_1 = beta_1, sigma_x = sigma_x, bias = bias,
             empirical_se = empirical_se, mse = mse, coverage = coverage)
}
```

Performance:

We run the simulation 1000 times to check performance.

```r
# Parameters
set.seed(123)         # Ensure reproducibility
beta_1_values <- c(0.5, 1, 5, 10)  # Test different true slopes
sigma_x_values <- c(1, 5, 10)     # Test different predictor variances
n_sim <- 1000                  # Number of simulations

# Run simulations for all combinations of beta_1 and sigma_x
results_list <- list()
for (beta_1 in beta_1_values) {
  for (sigma_x in sigma_x_values) {
    # Run simulation for each combination
```

```
    result <- run_simulation(beta_1 = beta_1, sigma_x = sigma_x, n_sim = n_sim)
    # Append result to the list
    results_list <- append(results_list, list(result))
  }
}

# Combine all results into a single data frame
results_all <- do.call(rbind, results_list)

# Print results
results_all %>%
  kable() %>%
  kable_styling()
```

beta_1	sigma_x	bias	empirical_se	mse	coverage
0.5	1	0.0029449	0.1052077	0.0110663	0.946
0.5	5	-0.0025117	0.0470816	0.0022208	0.942
0.5	10	0.0004195	0.0313370	0.0009812	0.954
1.0	1	0.0064656	0.0996773	0.0099674	0.955
1.0	5	0.0028416	0.0445622	0.0019919	0.951
1.0	10	-0.0009050	0.0328032	0.0010758	0.945
5.0	1	-0.0033658	0.1032405	0.0106593	0.949
5.0	5	-0.0022972	0.0449974	0.0020280	0.954
5.0	10	-0.0010782	0.0311281	0.0009692	0.963
10.0	1	-0.0004217	0.1038258	0.0107692	0.943
10.0	5	0.0016295	0.0456867	0.0020878	0.950
10.0	10	-0.0014375	0.0314246	0.0009886	0.957

As β_1 increases, we observe that the bias remains small across all scenarios, reflecting the generally unbiased nature of the OLS estimator. However, for larger values of β_1, such as 5.0 and 10.0, the bias becomes slightly negative. For instance, at $\beta_1 = 10.0$ and $\sigma_x = 1$, the bias is -0.0004, which suggests a minor underestimation introduced by the truncation rule ($p > 0.05$). This pattern aligns with the expectation that larger true effect sizes are more likely to encounter systematic underestimation due to the selection rule. Empirical standard error (SE) also increases slightly as β_1 grows, particularly noticeable when σ_x is fixed at smaller values. For example, when $\beta_1 = 0.5$ and $\sigma_x = 1$, the empirical SE is 0.1052, compared to 0.1038 when $\beta_1 = 10.0$. This indicates that as the true effect size grows, the variability in estimates increases, likely due to the larger absolute magnitude of the estimates. Mean squared error (MSE), which incorporates both bias and variability, also follows a slight increasing trend with larger β_1, reflecting the combined effects of increasing SE and minimal changes in bias.

Coverage, on the other hand, remains generally stable around the nominal level of 0.95 as β_1 increases, although minor deviations occur, likely attributable to simulation variability.

Turning to the effects of σ_x, increasing the variance of x significantly improves the model's performance across all metrics. For instance, at $\beta_1 = 0.5$, as σ_x increases from 1 to 10, the empirical SE decreases markedly from 0.1052 to 0.0313. This reduction in SE reflects the greater informational value provided by higher predictor variance, enabling more precise estimation of β_1. Similarly, MSE decreases substantially as σ_x grows, dropping from 0.0111 to 0.0010 in the same scenario. Bias, however, remains largely unaffected by changes in σ_x, staying consistently close to zero regardless of the variance. This aligns with the theoretical expectation that x's variance influences estimation precision rather than systematic error. Coverage shows slight but consistent improvement as σ_x increases, with values stabilizing closer to the nominal 0.95. For example, at $\beta_1 = 5.0$, coverage rises from 0.949 at $\sigma_x = 1$ to 0.963 at $\sigma_x = 10$. This suggests that higher predictor variance not only enhances precision but also ensures that confidence intervals are more likely to capture the true parameter value. The observed trends align with theoretical predictions, though the slight over-coverage at higher σ_x might reflect the impact of finite sample sizes or slight imbalances in confidence interval calculations.

Risk Score Simulation Study

In class, we went through the ADEMP framework to compare two methods for estimating integer risk score models. For this question, you will look at the computational part of designing a simulation study. Look at the simulate_data.R, experiments.R, and summarize_results.R scripts in the corresponding github repository: https://github.com/hjeglinton/RiskCD/tree/main/R

You should notice a few things

- We created a function to simulate our data and a function to store our data.
- We are storing the simulated data and underlying coefficients. We store these as .csv files but you could also store your files as .Rdata or .RDS files.
- The experiments script saves the results of our simulation as a csv file for future processing.
- Notice that we set a seed before generating our data and before comparing our models. This is because there is some underlying randomness in some of the applied methods.

Use what you observe to update your solution to the previous problem to better follow this format. You should have a data generation function that stores your generated data as well as a function to run that runs the methods and stores the results in an appropriate form.

```r
 # Function to generate data
gen_data <- function(param) {
  m <- param$m
  beta0 <- param$beta0
  beta1 <- param$beta1
  mu <- param$mu
  sd <- param$sd
  seed <- param$seed
  # set.seed(seed)
  x <- rnorm(m, mu, sd)
  y <- beta0 + beta1 * x + rnorm(m, 0, 1)
  data.frame(x = x, y = y)
}

# Save data
save_data <- function(data, params, file_dir) {
  if (!dir.exists(file_dir)) {
    dir.create(file_dir, recursive = TRUE)
  }
  data_file   <- file.path(file_dir, "data.csv")
  params_file <- file.path(file_dir, "params.csv")

  write.csv(data, data_file, row.names = FALSE)
  write.csv(params, params_file, row.names = FALSE)
}

# Run simulation
simul <- function(params, simul_name, num_simul) {

  results <- data.frame(
    id = integer(num_simul),
    beta1_est = numeric(num_simul),
    beta1_se = numeric(num_simul),
    beta1_lowerCI = numeric(num_simul),
    beta1_upperCI = numeric(num_simul),
    pval = numeric(num_simul),
    significant = logical(num_simul)
  )

  seed <- params$seed
  alpha <- params$alpha
  m <- params$m
```

```r
  df <- m - 2

  for (i in 1:num_simul) {
    params$seed <- seed + i
    data <- gen_data(params)
    file_dir <- file.path(simul_name, paste0("simul_no", i))
    save_data(data, params, file_dir)

    mdl <- lm(y ~ x, data = data)
    saveRDS(mdl, file.path(file_dir, "model.rds"))

    coefs <- summary(mdl)$coefficients
    write.csv(coefs, file.path(file_dir, "coef.csv"), row.names = TRUE)

    pval  <- coefs["x", "Pr(>|t|)"]
    if (pval < alpha) {
      beta1_est <- coefs["x", "Estimate"]
    } else {
      beta1_est <- 0
    }

    beta1_se <- coefs["x", "Std. Error"]

    quantile.t <- qt(1 - alpha / 2, df)
    beta1_lowerCI <- beta1_est - quantile.t * beta1_se
    beta1_upperCI <- beta1_est + quantile.t * beta1_se
    alpha <- params$alpha
    significant <- pval < alpha
    results$id[i] <- i
    results$beta1_est[i] <- beta1_est
    results$beta1_se[i] <- beta1_se
    results$beta1_lowerCI[i] <- beta1_lowerCI
    results$beta1_upperCI[i] <- beta1_upperCI
    results$pval[i] <- pval
    results$significant[i] <- significant
  }

  write.csv(results, file.path(simul_name, "simul_results.csv"), row.names = FALSE)
  return(results)
}

# Performance measures
```

```r
eval <- function(results, params, simul_name) {

  beta1_ast <- params$beta1
  # estimated beta_1
  beta1_est <- mean(results$beta1_est)
  # Bias
  bias <- beta1_est - beta1_ast
  # EmpSE
  EmpSE <- sd(results$beta1_est)
  # MSE
  MSE <- mean((results$beta1_est - beta1_ast)^2)
  # Coverage
  Coverage <- mean((results$beta1_lowerCI <= beta1_ast) & (results$beta1_upperCI >= beta1_ast
  # significant ratio
  significant_ratio <- mean(results$significant)

  eval_df <- data.frame(
    TrueBeta = beta1_ast,
    EstimatedBeta = beta1_est,
    Bias = bias,
    EmpSE = EmpSE,
    MSE = MSE,
    Coverage = Coverage,
    SignificantRatio = significant_ratio
  )
  write.csv(eval_df, file.path(simul_name, "performance.csv"),
            row.names = FALSE)
  return(eval_df)
}
```

```r
# Vary by variance of X
sd.vec <- c(1, 5, 10)
param.init <- data.frame(
  m = 100,
  beta0 = 0,
  beta1 = 1,
  alpha = 0.05,
  mu = 0,
  sd = 1,
  seed = 1
)
```

```r
eval_df <- data.frame()
for (val in sd.vec) {
  param <- param.init %>%
    mutate(sd = val)
  simul_name = sprintf("simulated/simul_xstd%d", val)
  results <- simul(param, simul_name, 100)
  eval_df <- eval(results, param, simul_name) %>%
    rbind(eval_df)
}
eval_df <- eval_df %>%
  mutate(X_Variance = sd.vec)

eval_df %>%
  kable() %>%
  kable_styling()
```

TrueBeta	EstimatedBeta	Bias	EmpSE	MSE	Coverage	SignificantRatio	X_Variance
1	0.9989425	-0.0010575	0.0114400	0.0001307	0.94	1	1
1	1.0026155	0.0026155	0.0207390	0.0004326	0.95	1	5
1	0.9879494	-0.0120506	0.0914144	0.0084182	1.00	1	10

```r
# Vary by Beta values
beta1.vec <- c(0.5, 1, 5, 10)
param.init <- data.frame(
  m = 100,
  beta0 = 0,
  beta1 = 1,
  alpha = 0.05,
  mu = 0,
  sd = 1,
  seed = 1
)

eval_df <- data.frame()
for (val in beta1.vec) {
  param <- param.init %>%
    mutate(beta1 = val)
  simul_name = sprintf("simulated/simul_beta%.1f", val)
  results <- simul(param, simul_name, 100)
  eval_df <- eval(results, param, simul_name) %>%
```

```
    rbind(eval_df)
}
eval_df <- eval_df %>%
  mutate(X_Variance = param$sd)

eval_df %>%
  kable() %>%
  kable_styling()
```

TrueBeta	EstimatedBeta	Bias	EmpSE	MSE	Coverage	SignificantRatio	X_Variance
10.0	10.0009464	0.0009464	0.1217409	0.0146735	0.92	1	1
5.0	4.9938045	-0.0061955	0.1021324	0.0103651	0.96	1	1
1.0	1.0055095	0.0055095	0.1015270	0.0102350	0.95	1	1
0.5	0.4895092	-0.0104908	0.1035280	0.0107209	0.95	1	1