KEN4258: Computational Statistics

Assignment 2

Aurélien Bertrand Bart van Gool Gaspar Kuper
Ignacio Cadarso Quevedo Nikola Prianikov

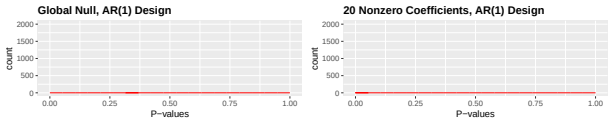
March 2024

Assignment 2

Link to our GitHub repository: https://github.com/nprianikov/compstats

1) Reproduce Figure 1 from (Candès et al. 2018).

```
library(tibble)

#set.seed(101)

n <- 500
p <- 200

# Generate X as n AR(1) time-series of size p
generate_AR1 <- function(n, p) {
  X <- replicate(n, arima.sim(n=p, list(0.5)))
  return(t(X))
}

# Function to generate responses, fit a logistic regression model and return individual p-values
generate_responses <- function(n, p, prob = NULL) {
  X <- generate_AR1(n=n, p=p)
  if (is.null(prob)) {
    prob <- plogis(0.08*rowSums(X[, 2:22]))
  }
  Y <- rbinom(n, 1, prob)

  fit <- glm(Y ~ X, family=binomial(link="logit"))

  p_values <- summary(fit)$coef[, "Pr(>|z|)"][2] # Take p-values for beta_1 only
  return(p_values)
}

n_sim <- 1
df_plot <- tibble(
  p_values_1 = c(replicate(n_sim, generate_responses(n=n, p=p, prob=0.5))),
  p_values_2 = c(replicate(n_sim, generate_responses(n=n, p=p)))
)
```



2) What is the problem that Figure 1 tries to illustrate?

Figure 1 illustrates the p-values for coefficient β_1 in two GLMs for two different systems. The figure shows, for each system, a histogram of the distribution of the β_1 p-value over $10,000$ replications of the experiment. The systems are given by the following functions:

(1) $Y|X_1, \ldots, X_p \sim \text{Bernoulli}(0.5)$

(2) $Y|X_1, \ldots, X_p \sim \text{Bernoulli}(\text{logit}(0.08(X_2 + \ldots + X_{21})))$

Where $(X_1, ..., X_p)$ are themselves random variables generated by an AR(1) time series with AR coefficient 0.5, and $p = 200$. In other words, the results of the first simulation are completely independent from the 200 values of X_i while the second simulation is only dependent on $(X_2, ..., X_{21})$. Neither system is influenced in any way by the predictor X_1 nor its coefficient β_1.

The figure shows that in both simulations, we see a significant amount of low p-values. In fact, for (1) we have 16.89% of p-values ≤ 16.89, while for (2) it is 19.17%. In GLMs, the p-values of a coefficient indicate the importance of a coefficient to the model. Low p-values are linked to coefficients that are significant to the model, and are therefore also significantly different from 0. This does not seem to be reflected in the figures, however, where we see that the β_1 coefficient is often having low p-values even though it has no impact at all on the underlying system that the model is trying to fit.

The problem can arise for multiple reasons in high-dimensional datasets, especially when the number of predictors is large relative to the amount of observations. One such reason is the multiple comparisons problem which occurs simply because so many hypotheses are being tested. In simulation (1) for example, where we have 200 predictors with no relation to the outcome, we can still expect to identify 10 important variables with a confidence of 95%. Another reason could be due to the model overfitting to the outcome variable. Even though predictors may not have any relationship with the outcome, they may still be able to help the model fit to this specific set of outcomes. This problem is especially common when we have a relatively small amount of observations.

To correct this problem, we will need to implement a method to decrease the false discovery rate such that we are less likely to label a predictor as being significant.

3) Propose a solution to address the problem.

```r
# draw a new sample from the conditional distribution of Xj | X-j using a random number generator
# Not sure if this is the correct way to sample
simulate_Xj_given_Xj <- function(X, j) {
  Xj <- X[, j]
  X_minus_j <- X[, -j, drop=FALSE]

  # Fit a model to predict Xj from the rest of the data
  model <- glm(Xj ~ X_minus_j - 1, family="gaussian")
  predicted_Xj <- predict(model, newdata=list(X_minus_j=X_minus_j))

  residuals_sd <- sd(resid(model))

  # Generate new samples for Xj with added randomness
  new_Xj_samples <- predicted_Xj + rnorm(length(predicted_Xj), mean=0, sd=residuals_sd)

  return(new_Xj_samples)
}

# Feature importance statistic function to test whether Xj and Y are conditionally independent.
# Using absolute value of an estimated coefficient
# Source paper uses abs. value of Lasso-estimated coefficient
compute_Tj <- function(X, y) {
  fit <- glm(y ~ X - 1, family=binomial(link="logit"))
  coef <- coef(fit)
```

```
  return(abs(coef))
}

# Conditional Randomization Test for a specific j
conditional_randomization_test <- function(X, y, j, K = 200) {
  original_Tj <- compute_Tj(X, y)[j]
  greater_count <- 0

  for (k in 1:K) {
    # Create a new data matrix by simulating the jth column and keeping the remaining columns the sa
    X_simulated <- X
    X_simulated[, j] <- simulate_Xj_given_Xj(X, j)

    # Compute Tj for the simulated data
    simulated_Tj <- compute_Tj(X_simulated, y)[j]
    # Increment count if simulated Tj is greater than or equal to the original Tj
    if (simulated_Tj >= original_Tj) {
      greater_count <- greater_count + 1
    }
  }
  # Calculate p-value
  p_value <- (1 + greater_count) / (K + 1)
  return(p_value)
}
```

```
# Conditional Randomization Test for entire matrix X
conditional_randomization_test_all <- function(X, y, K = 1) {
  p_values <- numeric(ncol(X))
  for (j in 1:ncol(X)) {
    p_values[j] <- conditional_randomization_test(X, y, j, K)
  }
  return(p_values)
}

X <- generate_AR1(n=n, p=p)
Y <- rbinom(n, 1, 0.5)
p_values <- conditional_randomization_test_all(X, Y)
print(p_values)
```

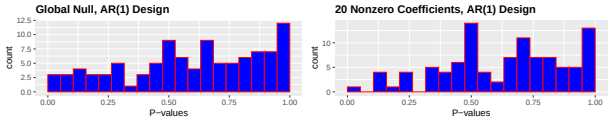
```
##   [1] 0.5 0.5 0.5 1.0 0.5 1.0 1.0 1.0 1.0 0.5 0.5 1.0 0.5 0.5 0.5 0.5 1.0 0.5
##  [19] 0.5 1.0 1.0 1.0 0.5 1.0 0.5 1.0 1.0 1.0 0.5 1.0 0.5 1.0 0.5 1.0 0.5 1.0
##  [37] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.5 1.0 0.5 1.0 0.5 1.0 1.0 1.0 0.5 1.0 0.5
##  [55] 0.5 0.5 1.0 1.0 0.5 1.0 0.5 0.5 0.5 0.5 1.0 1.0 1.0 1.0 0.5 1.0 0.5 1.0
##  [73] 1.0 0.5 1.0 1.0 1.0 0.5 1.0 0.5 1.0 1.0 1.0 1.0 1.0 1.0 0.5 1.0 1.0 1.0
##  [91] 0.5 1.0 1.0 1.0 1.0 1.0 0.5 0.5 0.5 1.0 1.0 1.0 0.5 1.0 1.0 0.5 1.0 0.5
## [109] 0.5 1.0 1.0 0.5 0.5 1.0 0.5 1.0 1.0 0.5 0.5 1.0 1.0 1.0 1.0 0.5 1.0 0.5
## [127] 1.0 1.0 1.0 1.0 1.0 0.5 1.0 0.5 0.5 1.0 0.5 1.0 1.0 1.0 1.0 1.0 0.5 0.5
## [145] 0.5 1.0 0.5 1.0 0.5 1.0 1.0 0.5 1.0 0.5 0.5 1.0 0.5 1.0 0.5 0.5 0.5 1.0
## [163] 0.5 1.0 0.5 1.0 0.5 1.0 0.5 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
## [181] 0.5 0.5 0.5 1.0 1.0 1.0 1.0 1.0 0.5 1.0 1.0 1.0 0.5 0.5 0.5 1.0 1.0 0.5
## [199] 1.0 0.5
```

4) Show that your solution fixes the problem.

The code below is an attempt (thus not sure if it is fully correct) to show how CRT fixes issue with the original plot.

```
generate_responses_crt <- function(n, p, prob = NULL) {
  X <- generate_AR1(n=n, p=p)
```

```
  if (is.null(prob)) {
    prob <- plogis(0.08*rowSums(X[, 2:22]))
  }
  Y <- rbinom(n, 1, prob)
  # assuming that we want to be confident in showing that p-values are <= 0.05
  # set K = 20 (reciprocal of 0.05)
  # we are interested in running CRT for the first feature
  p_values <- conditional_randomization_test(X, Y, j=1, K=20)
  return(p_values)
}

fixed_p_values_1 <- replicate(100, generate_responses_crt(n=n, p=p, prob=0.5))
fixed_p_values_2 <- replicate(100, generate_responses_crt(n=n, p=p))
```



5) Find a real dataset and apply your method.