Resampling Techniques Assignment

Subhrajyoty Roy (MB1911)

2020-05-31

/ Introduction

This assignment concerns with the implementation of Bootstrap Approximation to Neareast Neighbour Estimates and finding confidence interval for the estimates using Hybrid Bootstrap technique. This assignment performs some necessary simulations to understand how the approximate confidence interval for the nearest neighbour estimate using theory of asymptotic normality compares that of obtained through hybrid bootstrap resampling process, and in a way, it tries to verify the results reported by `Shao et. al.[1]` and `Dikta[2]`.

/ The Theory

// Problem

Let us assume, (x_i, y_i), where $i = 1, 2, \dots n$ be a i.i.d. random sample where x_i's may be fixed or random, and the y_i is given as;

$$y_i = f(x_i) + \epsilon_i$$

where $f(\cdot)$ is an unknown function and ϵ_i's are error which are assumed to satisfy two basic properties.

1. $\mathbb{E}(\epsilon_i \mid x_i) = 0$ if x_i is random, or $\mathbb{E}(\epsilon_i) = 0$ if x_i's are non random, but a fixed quantity.
2. If x_i's are fixed, $\text{Var}(\epsilon_i) = \sigma^2$, where σ^2 does not depend on the index i, and is unknown.

The goal is to find an estimate or confidence interval for the value of the function f at an unknown (or previously unseen) point x_0 i.e. to find $f(x_0)$, where x_0 is assumed to within the range of observed x_i's. For this reason, without loss of generality, x_i's can be bounded in a compact set like $[0, 1]$ or $[-1, 1]$ after proper normalization.

// Nearest Neighbour Regression Estimate

The nearest neighbour regression estimate of $f(x_0)$ is basically obtained a weighted average of the y_i's where weights are generally such that only the x_i's which are nearest (or upto k-th closest neighbour) to the desired point x_0 is taken into consideration.

Assume, (X_i, Y_i)'s are iid samples, such that $f(X) := \mathbb{E}(Y \mid X)$. Then, the smoothed nearest neighbour regression type estimate based on n observed datapoints is simply:

$$\begin{aligned}\widehat{f_n(x_0)} &= \frac{1}{n\lambda_n}\sum_{i=1}^{n} Y_i K\left(\frac{1}{\lambda_n}(F_n(x_0) - F_n(X_i))\right) \\ &= \frac{1}{\lambda_n}\int yK\left(\frac{1}{\lambda_n}(F_n(x_0) - F_n(x))\right) dH_n(x,y)\end{aligned}$$

where $H_n(x, y)$ is empirical cdf of the joint distribution of (X, Y) and F_n is the empirical cdf of distribution of X.

However, according to `Dikta[2]`, the above estimate can be revised in the following way to give birth to a superior regression estimate, which is better in small sample situations and has the same limiting distribution as $\widehat{f_n(x_0)}$.

$$\widehat{f_{n0}(x_0)} = \widehat{f_n(x_0)}\left[\frac{1}{\lambda_n}\int K\left(\frac{1}{\lambda_n}(F_n(x_0) - F_n(x))\right) dF_n(x)\right]^{-1}$$

// Normal Approximation to Confidence Interval Estimates

Combining the remarks made by `Dikta[2]` and `Stute[3]`, the $100(1-\alpha)\%$ confidence interval estimate to $f(x_0)$ using nearest neighbour regression approach can be obtained as;

$$\left[\widehat{f_{n0}(x_0)} - z_{1-\alpha/2}\frac{\widehat{\sigma_0}}{\sqrt{n\lambda_n}}, \widehat{f_{n0}(x_0)} + z_{1-\alpha/2}\frac{\widehat{\sigma_0}}{\sqrt{n\lambda_n}}\right]$$

where z_α is the α-th quantitle of standard normal distribution, and

$$\widehat{\sigma_0^2} = \left[\left\{\int y^2 K\left(\frac{F_n(x_0) - F_n(x)}{\lambda_n}\right) dH_n(x,y)\right\}\right. \\ \left.\left\{\int K\left(\frac{F_n(x_0) - F_n(x)}{\lambda_n}\right) dF_n(x)\right\}^{-1} - \widehat{f_{n0}(x_0)}^2\right]\int K^2(u)du$$

For the simulation purpose, here we use the **Epanechnikov Kernel**, namely;

$$K(x) = \frac{3}{4}(1 - x^2)\mathbf{1}_{[-1,1]}(x)$$

In that case, $\int K^2(u)du = 3/5$, which is to be used to obtain $\widehat{\sigma_0^2}$. The following `R` function computes the confidence interval given the data for this kernel.

```r
estimate.norm <- function(x, data, lambda = 0.35, conf = 0.9) {
    f <- ecdf(data$x)
    norm.x <- (f(x) - f(data$x)) / lambda      # apply ecdf
    kern.weights <- (3/4) * (1 - norm.x^2) * ifelse(abs(norm.x) > 1, 0, 1)

    estimate <- weighted.mean(data$y, kern.weights)

    var_est <- (weighted.mean(data$y^2, kern.weights) - estimate^2) * (3/5)    # (3/5) =
  integral(K(u)du)
    len_est <- qnorm((1 - conf)/2, lower.tail = FALSE) * sqrt(var_est) / sqrt(nrow(data) *
  lambda)

    result <- c(estimate - len_est, estimate + len_est)  # the CI
    return(result)
}
```

Hybrid Bootstrap Confidence Interval Estimates

For Bootstrap based confidence interval estimates, given the observed data $(X_1, Y_1), (X_2, Y_2) \dots (X_n, Y_n)$, we compute the empirical cdf of (X, Y) as H_n, and obtain bootstrap resamples (X_i^*, y_i^*) from the bivariate cdf H_n, and obtain corresponding empirical cdf of bootstrap resamples X^*'s as F_n^*. Then, the boostrapped version of nearest neighbour estimates are given by;

$$\widehat{f_n^*(x_0)} = \frac{1}{\lambda_n} \int yK\left(\frac{1}{\lambda_n}(F_n^*(x_0) - F_n^*(x))\right) dH_n^*(x, y)$$

and

$$\widehat{f_{n0}^*(x_0)} = \widehat{f_n^*(x_0)} \left[\frac{1}{\lambda_n} \int K\left(\frac{F_n^*(x_0) - F_n^*(x)}{\lambda_n}\right) dF_n^*(x)\right]^{-1}$$

Based on these estimates, we compute the $\alpha/2$-th and $(1-\alpha/2)$-th quantiles of the distribution function of $\sqrt{n\lambda_n}\left(\widehat{f_{n0}^*(x_0)} - \widehat{f_{n0}(x_0)}\right)$ which can be used to approximate the distribution function of $\sqrt{n\lambda_n}\left(\widehat{f_{n0}(x_0)} - f(x_0)\right)$. Instead of computing the exact quantiles from all bootstrap samples, we can simply use the method of Monte Carlo to simulate bootstrap resamples and use the quantiles obtained from those resamples as an approximation. Finally, the $100(1-\alpha)\%$ hybrid bootstrap confidence interval is given as;

$$\left[2\widehat{f_{n0}(x_0)} - (\widehat{f_{n0}^*(x_0)})_{(B(1-\alpha/2))}, 2\widehat{f_{n0}(x_0)} - (\widehat{f_{n0}^*(x_0)})_{(B(\alpha/2))}\right]$$

where the index in brackets denote the order statistics and B is the number of Bootstrap resamples.

The following code obtains bootstrap confidence interval given the input data.

```r
estimate.boot <- function(x, data, seed = 1234, B = 1e3, lambda = 0.35, conf = 0.9) {
    set.seed(seed)
    boot_samples <- sample(1:1e9, size = B)   # generate some seeds to be used to generate
  data

    boot_ests <- numeric(B)
    for (b in 1:B) {
        index <- sample(1:nrow(data), size = nrow(data), replace = TRUE)
        boot_data <- data[index, ]

        f <- ecdf(boot_data$x)
        norm.x <- (f(x) - f(boot_data$x)) / lambda     # apply ecdf
        kern.weights <- (3/4) * (1 - norm.x^2) * ifelse(abs(norm.x) > 1, 0, 1)

        boot_ests[b] <- weighted.mean(boot_data$y, kern.weights)
    }

    # now obtain original f(hat)_0(x_0) from the original data
    f <- ecdf(data$x)
    norm.x <- (f(x) - f(data$x)) / lambda     # apply ecdf
    kern.weights <- (3/4) * (1 - norm.x^2) * ifelse(abs(norm.x) > 1, 0, 1)
    f_n0 <- weighted.mean(data$y, kern.weights)

    result <- 2 * f_n0 - quantile(boot_ests, probs = c(1 - (1-conf)/2, (1-conf)/2 ) )  #
  obtain CI

    return(result)
}
```

Simulation

In the simulation study as described in `Dikta[2]`, we consider two models.

1. **Model 1:** $X \sim \text{Uniform}(-1, 1)$, $\epsilon \sim \text{Uniform}(-0.5, 0.5)$, both being independent to each other. Finally,

$$Y = 5X^2 + 7X + \epsilon$$

2. **Model 2:** X, ϵ are as generated in model 1. However,

$$Y = 5X^2 + 7X + (X + 1)\epsilon$$

Clearly, in model 2, homoscadasticity assumption is violated.

We use the following specifications:

1. Sample size, $n = 50$.
2. Number of bootstrap resamples, $B = 1000$.
3. The intended point at which estimates are needed, $x_0 \in \{-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7\}$
4. The optimal bandwidth $\lambda_n := (n\ln(n))^{-(1/5)} \approx 0.35$
5. $\alpha = 0.1$, i.e. we wish to construct 90% confidence intervals.

In each of the scenario, we generate 100 simulated data from Model 1 (or Model 2), then obtain the normal and bootstrap confidence intervals for each data. Since, we have the actual value of $f(x) = 5x^2 + 7x$, at all intended points, it is checked whether the obtained confidence interval contains the actual observation. Then, the proportion of times the confidence interval contains the true estimate is reported as an approximate confidence coefficient, and the average length of the confidence intervals are also obtained.

The following code simulates the data from the given model.

```
sim.model <- function(n, seed = 1234, model = 1) {
    set.seed(seed)
    x <- runif(n, min = (-1), max = 1)
    error <- runif(n, min = (-1/2), max = (1/2))
    if (model == 1) {
        y <- 5 * x^2 + 7 * x + error
    } else {
```

```r
        y <- 5 * x^2 + 7 * x + ((x + 1) * error)
    }
    return(data.frame(x = x, y = y))
}
```

Now, we create the following function, which performs the 100 simulations for us and yields the result.

```r
library(pbapply)   # to use a progress bar
library(parallel)  # to use parallel programming to speed up the computation

nCores <- detectCores()

get.result <- function(x0, seed = 1234, model = 1, type = "norm") {
    # x0 is the point at which are estimate is needed
    # seed is something to make the results reproducible
    # model is whether model 1 or model 2
    # type = "norm" for normal approximation or "boot" for bootstrap approximation

    # set the specifications
    n <- 50
    lambda <- (n * log(n))^(-1/5)
    B <- 1000
    sim_size <- 100
    conf <- 0.9

    set.seed(seed)
    data_seeds <- sample(1:1e9, size = sim_size)   # generate some seeds to be used to
  generate data

    # perform the simulations
    res <- pbsapply(X = 1:sim_size, FUN = function(i) {
        data <- sim.model(n, seed = data_seeds[i], model = model)

        if (type == "norm") {
            ci <- estimate.norm(x = x0, data = data, lambda = lambda, conf = conf)
        }
        else if (type == "boot") {
            ci <- estimate.boot(x = x0, data = data, seed = data_seeds[i], B = B, lambda =
  lambda, conf = conf)
        }
        else {
          stop("Type must be either 'norm' for normal approximation or 'boot' for bootstrap
  approximation")
        }

        true_val <- 5 * x0^2 + 7 * x0   # true f(x0)

        cp <- (ci[1] <= true_val) & (true_val <= ci[2])   # coverage indicator
        len <- ci[2] - ci[1]    # length of the confidence interval

        return(c(cp, len))

    }, cl = nCores)   # run simulations parallely

    # now actual estimates
    res1 <- as.numeric(rowMeans(res))

    return(c(CP = res1[1], WL = res1[2]))
}
```

Simulation Results

I use my roll number *1911* as the seed to make the exact result reproducible and replicable.

Model 1 with Normal Approximation

We store the results for model 1 with normal approximation confidence intervals for the intended choices of x_0, in a dataframe.

```
tabdf1 <- data.frame(x0 = c(-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7), CP = rep(NA, 7), WL =
  rep(NA, 7))

for (i in 1:7) {
    tmp <- get.result(x0 = tabdf1$x0[i], seed = 1911, model = 1, type = "norm")
    tabdf1$CP[i] <- tmp[1]    # save the result
    tabdf1$WL[i] <- tmp[2]
}

# finally print the dataframe
knitr::kable(tabdf1)
```

x0	CP	WL
-0.7	0.05	0.3411435
-0.4	0.12	0.7210817
-0.2	0.63	1.0520068
0.0	0.80	1.4390581
0.2	0.92	1.7837285
0.4	0.97	2.0172443
0.7	0.13	1.9148401

Model 1 with Bootstrap Approximation

We store the results for model 1 with bootstrap approximation confidence intervals for the intended choices of x_0, in a dataframe.

```
tabdf2 <- data.frame(x0 = c(-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7), CP = rep(NA, 7), WL =
  rep(NA, 7))

for (i in 1:7) {
```

```r
    tmp <- get.result(x0 = tabdf2$x0[i], seed = 1911, model = 1, type = "boot")
    tabdf2$CP[i] <- tmp[1]   # save the result
    tabdf2$WL[i] <- tmp[2]
}

# finally print the dataframe
knitr::kable(tabdf2)
```

x0	CP	WL
-0.7	0.38	0.5467567
-0.4	0.34	0.7695989
-0.2	0.59	0.9953990
0.0	0.73	1.2893649
0.2	0.70	1.4883735
0.4	0.78	1.4801597
0.7	0.34	1.9646186

// Model 2 with Normal Approximation

We store the results for model 2 with normal approximation confidence intervals for the intended choices of x_0, in a dataframe.

```r
tabdf3 <- data.frame(x0 = c(-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7), CP = rep(NA, 7), WL =
  rep(NA, 7))

for (i in 1:7) {
    tmp <- get.result(x0 = tabdf3$x0[i], seed = 1911, model = 2, type = "norm")
    tabdf3$CP[i] <- tmp[1]   # save the result
    tabdf3$WL[i] <- tmp[2]
}

# finally print the dataframe
knitr::kable(tabdf3)
```

x0	CP	WL
-0.7	0.00	0.3012172
-0.4	0.06	0.7092667
-0.2	0.62	1.0478438
0.0	0.80	1.4404293

x0	CP	WL
0.2	0.92	1.7876208
0.4	0.97	2.0221666
0.7	0.12	1.9218372

Model 2 with Bootstrap Approximation

We store the results for model 2 with bootstrap approximation confidence intervals for the intended choices of x_0, in a dataframe.

```r
tabdf4 <- data.frame(x0 = c(-0.7, -0.4, -0.2, 0, 0.2, 0.4, 0.7), CP = rep(NA, 7), WL =
  rep(NA, 7))

for (i in 1:7) {
    tmp <- get.result(x0 = tabdf4$x0[i], seed = 1911, model = 2, type = "boot")
    tabdf4$CP[i] <- tmp[1]   # save the result
    tabdf4$WL[i] <- tmp[2]
}

# finally print the dataframe
knitr::kable(tabdf4)
```

x0	CP	WL
-0.7	0.30	0.5091329
-0.4	0.37	0.7549028
-0.2	0.59	0.9911759
0.0	0.71	1.2917294
0.2	0.70	1.4900818
0.4	0.80	1.4838090
0.7	0.33	1.9786732

Conclusion

The outputs from all previous setups are combined to create the final table as follows:

```r
library(kableExtra)

kable(cbind(tabdf1, tabdf2[, -1], tabdf3[, -1], tabdf4[, -1]), format = "html") %>%
  add_header_above(c(" ", "Normal Approx." = 2, "Bootstrap Approx." = 2,
                     "Normal Approx." = 2, "Bootstrap Approx." = 2)) %>%
  add_header_above(c(" ", "Model 1" = 4, "Model 2" = 4))
```

	Model 1				Model 2			
	Normal Approx.		Bootstrap Approx.		Normal Approx.		Bootstrap Approx.	
x0	CP	WL	CP	WL	CP	WL	CP	WL
-0.7	0.05	0.3411435	0.38	0.5467567	0.00	0.3012172	0.30	0.5091329
-0.4	0.12	0.7210817	0.34	0.7695989	0.06	0.7092667	0.37	0.7549028
-0.2	0.63	1.0520068	0.59	0.9953990	0.62	1.0478438	0.59	0.9911759
0.0	0.80	1.4390581	0.73	1.2893649	0.80	1.4404293	0.71	1.2917294
0.2	0.92	1.7837285	0.70	1.4883735	0.92	1.7876208	0.70	1.4900818
0.4	0.97	2.0172443	0.78	1.4801597	0.97	2.0221666	0.80	1.4838090
0.7	0.13	1.9148401	0.34	1.9646186	0.12	1.9218372	0.33	1.9786732

From the table above, we see the following:

1. For both ends when the x_0 is close to (-1) or 1, the normal confidence coefficient performs extremely bad, and comparatively bootstrap confidence interval performs much better.
2. For x_0 close to 0, the bootstrap confidence coefficient is lower than confidence coefficient of the interval obtained using normal approximation.
3. The confidence intervals obtained using bootstrap estimates are generally shorter in length than the ones obtained using normal approximation.

/ References

1. **The Jackknife and Bootstrap** - Jun Shao and Donsheng Tu, Springer, Verlag New York (1995).
2. **Bootstrap Approximation to Nearest Neighbour Regression Function Estimates** - Gerhard Dikta, Journal of Multivariate Analysis, Vol - 32, pp 213-229 (1990).
3. **Asymptotic normality of nearest neighbor regression function estimates** - Stute W., Annals of Statistics, Vol - 12, pp 917-926 (1984).

/ THANK YOU