Homework 3

Costa Stavrianidis

2022-10-31

Problem 1

Suppose that Y1,...,Yn is a simple random sample from a normal distribution with mean μ and variance $\sigma^2 > 0$

- **Construct an exact two-sided $1-\alpha$ confidence interval estimator for the population mean μ (hint use the pivot method from BIOS 704; See 704 lecture notes and Bain and Engelhard)**
- **Write a function to implement the previous estimator**
- **Outline (in writing) a simulation study for empirical assessment of coverage probability using N simulation replicates for the confidence interval estimator**
- **Exemplify the use of your simulation study using n=25 and n=50 (sample sizes), $\mu = 1$ (population mean), $\sigma = 0.25$ (population standard deviation), $\alpha = 0.05$ (level of significance) and N=10,000 (simulation replicates).**
- **Write a function to implement a two-sided $1-\alpha$ bootstrap quantile estimator for the population mean μ using B bootstrap replicates (see BIOS 823 lecture notes)**
- **Outline (in writing) a simulation study for empirical assessment of coverage probability using N simulation replicates for the bootstrap interval estimator**
- **Exemplify the use of your simulation study using n=25 and n=50 (sample sizes), $\mu = 1$ (population mean), $\sigma = 0.25$ (population standard deviation), $\alpha = 0.05$ (level of significance), B=1000 (boostrap replicates) and N=10,000 (simulation replicates).**

You may use Python or Rcpp to implement these functions and simulation studies. The two requested outlines should be detailed technical accounts (how would you outline your proposed approach to your advisor or supervisor?).

Two-sided confidence interval:

$$(\bar{y} - t_{1-\alpha/2} \cdot s/\sqrt{n}, \bar{y} + t_{1-\alpha/2} \cdot s/\sqrt{n})$$

Function to implement previous estimator:

```
conf <- function(x, alpha) {
  c1 <- mean(x) - qt(1-alpha/2, length(x)-1) * sd(x) / sqrt(length(x))
  c2 <- mean(x) + qt(1-alpha/2, length(x)-1) * sd(x) / sqrt(length(x))
  return(c(c1, c2))
}

# Testing function with randomly generated random sample
x <- rnorm(1000, 50, 3)
conf(x, .05)
```

```
## [1] 49.77822 50.15798
```

Simulation study outline for confidence interval estimator: The first step is to simulate random samples from a population with a known mean and standard deviation. The size of these random samples can change as you repeat the study. For each sample you take, calculate the exact confidence interval using the function written above. Since the population mean is known, we can then compute the proportion of our calculated confidence interval that contain this true population mean. The proportion of intervals that contain this mean is the estimate for our coverage probability in the study.

Implementing simulation study:

```
interval_performance <- function(n, runs) {

  exact_cov <- rep(NA, runs)

  for (i in 1:runs) {

    x <- rnorm(n, 1, 0.25)

    exact_int <- conf(x, 0.05)

    exact_cov[i] <- between(1, exact_int[1], exact_int[2])

  }

  exact_cov_prob <- sum(exact_cov) / runs

  return(exact_cov_prob)

}

interval_performance(25, 10000)
```

```
## [1] 0.9501
```

```
interval_performance(50, 10000)
```

```
## [1] 0.9484
```

Function to implement bootstrap quantile estimator:

```
boot_conf <- function(x, alpha, B, n) {
  boot <- rep(NA, B)
  for (i in 1:B) {
    boot[i] = mean(sample(x, n, replace = T))
  }
  return(c(quantile(boot, alpha/2), quantile(boot, 1 - alpha/2)))
}

# Testing function on randomly generated random sample
boot_conf(rnorm(1000, 50, 3), .05, 1000, 20)
```

```
##     2.5%    97.5%
## 48.74031 51.54931
```

Simulation study outline for bootstrap interval estimator: First, we will simulate a random sample of a given size from a given distribution. To create the bootstrapped quantile estimator, we will sample with replacement from the simulated random sample a certain amount of times (defined in the bootstrap function as B), and then take the 2.5% and 97.5% quantiles from the means of all of these samples. This will give us a bootstrapped interval estimator for the population mean from B bootstrap replicates. In the study, we

can replicate this as many times as we want, and then calculate a final proportion of bootstrapped interval estimators that contain the true population mean. This proportion will be our empirical coverage probability.

Implementing simulation study:

```
bootstrap_performance <- function(n, runs) {

  boot_cov <- rep(NA, runs)

  for (i in 1:runs) {

    x <- rnorm(n, 1, 0.25)

    boot_int <- boot_conf(x, 0.05, 1000, n)

    boot_cov[i] <- between(1, boot_int[1], boot_int[2])

  }

  boot_cov_prob <- sum(boot_cov) / runs

  return(boot_cov_prob)

}

bootstrap_performance(25, 10000)
```

```
## [1] 0.9337
```

```
bootstrap_performance(50, 10000)
```

```
## [1] 0.9392
```

Problem 2

Consider the simple linear regression model where Y1,. . . ,Yn are independent and each Yi follows a normal distribution with mean $\alpha + \beta x_i$ and common variance $\sigma^2 > 0$. Needless to say, this can be reformulated as a GLM.

- **Write out the likelihood function for $(\alpha, \beta, \sigma^2)$ (use results from notes)**
- **Write out the gradient vector**
- **Find the maximum-likelihood estimator for $(\alpha, \beta, \sigma^2)$**
- **Use pytorch or tensorflow to find the maximum likelihood estimates for the toy example**

Likelihood function for $(\alpha, \beta, \sigma^2)$:

$$L(\alpha, \beta, \sigma^2) = \frac{1}{\sqrt{(2\pi\sigma^2)^n}} exp[-\frac{1}{2\sigma^2}||\tilde{y} - \alpha + \beta X||^2]$$

$$Log - likelihood(\alpha, \beta, \sigma^2) = log(L(\alpha, \beta, \sigma^2)) = -\frac{n}{2}log(2\pi) - \frac{n}{2}log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \alpha - \beta)$$

Gradient vector:

$$\begin{pmatrix} \frac{\partial}{\alpha}(LL(\alpha, \beta, \sigma^2)) \\ \frac{\partial}{\beta}(LL(\alpha, \beta, \sigma^2)) \\ \frac{\partial}{\sigma^2}(LL(\alpha, \beta, \sigma^2)) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i - \alpha - \beta) \\ \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i y_i - \alpha x_i - \beta x_i^2) \\ -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2 \end{pmatrix}$$

Maximum likelihood estimator for $(\alpha, \beta, \sigma^2)$:

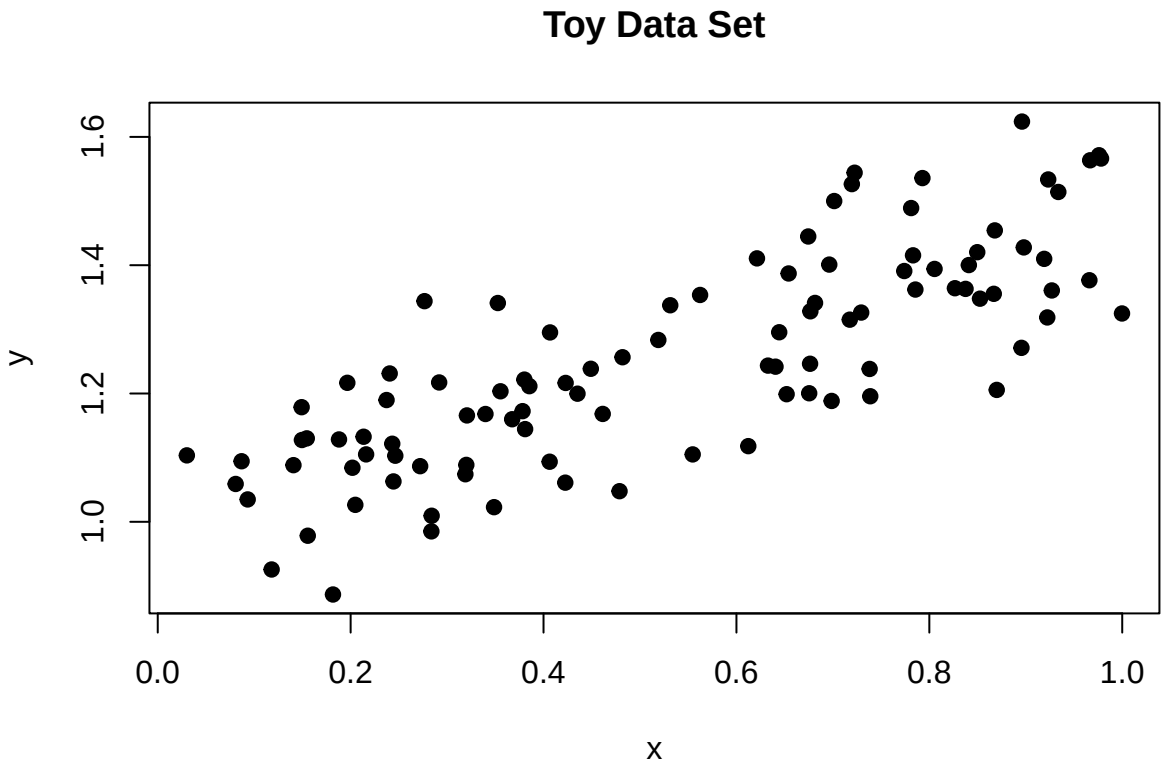
$$MLE(\alpha, \beta, \sigma^2) = \begin{pmatrix} \frac{\partial}{\alpha}(LL(\alpha, \beta, \sigma^2)) = 0 \\ \frac{\partial}{\beta}(LL(\alpha, \beta, \sigma^2)) = 0 \\ \frac{\partial}{\sigma^2}(LL(\alpha, \beta, \sigma^2)) = 0 \end{pmatrix} = \begin{pmatrix} \hat{\alpha} = \bar{y} - \hat{\beta}\bar{x} \\ \hat{\beta} = \frac{sxy}{s_x^2} \\ \hat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\alpha} - \hat{\beta}x_i)^2 \end{pmatrix}$$

Finding MLE using PyTorch/Tensorflow for example:

```
set.seed(1281)
n <- 100
x <- runif(n, 0, 1)

y <- 1 + 0.5 *x + rnorm(n, 0, 0.1)

plot(x, y, pch = 19, main = "Toy Data Set")
```




```
import pandas as pd
import numpy as np
import torch
import math
from torch.autograd import Variable

np.random.seed(1281)
n = 100
x = np.random.uniform(size=n)
y = 1 + 0.5 * x + np.random.normal(0, 0.1, size=n)

X = Variable(torch.from_numpy(x)).type(torch.FloatTensor)
Y = Variable(torch.from_numpy(y)).type(torch.FloatTensor)
alpha = torch.tensor(np.array([1]), dtype=torch.float64, requires_grad=True)
```

```
beta = torch.tensor(np.array([0.5]), dtype=torch.float64, requires_grad=True)
sigma = Variable(torch.rand(1), requires_grad=True)

learning_rate = 0.0001
OPT_OBJ = torch.optim.SGD([alpha, beta, sigma], lr = learning_rate)

for t in range(10000):
    NLL = (n/2) * math.log(2*math.pi * sigma.pow(2)) + (1/2*sigma.pow(2)) * sum(((Y - alpha - X*beta).p
    OPT_OBJ.zero_grad()
    NLL.backward()
    OPT_OBJ.step()

torch_alpha = alpha.data.numpy()[0]
torch_beta = beta.data.numpy()[0]
torch_var = (sigma.data.numpy()[0])**2

print(torch_alpha, torch_beta, torch_var)
```

```
## 0.995718282947227 0.4924918200895393 0.020682418733095398
```

Compare results to closed-form expressions from class notes:

```
alpha = np.mean(y) - beta * np.mean(x)
print(torch_alpha - alpha)
```

```
## tensor([-0.0002], dtype=torch.float64, grad_fn=<RsubBackward1>)
```

```
beta = np.std(x) * x * y / np.std(x)**2
print(torch_beta - beta)
```

```
## [-0.564186   -1.20506844 -2.55937948 -0.25238046  0.16558181 -3.63695225
##   0.39764017 -2.2061443  -4.22207892 -2.02574597 -0.28570057 -3.91513686
##  -4.22096112 -3.73558433 -1.38504577 -3.34026151 -3.82132143 -1.67601378
##  -3.46069817 -0.04385956 -0.24289797 -2.35564219 -1.39844913 -3.94166635
##  -0.93844514 -1.8716566  -1.92098987  0.43894491  0.41334742 -2.51895737
##  -1.91412128 -2.75708008  0.01582276 -1.78702794 -2.74681746 -1.57353297
##  -4.35676155  0.18386425 -0.73184176 -2.297585   -0.79007652 -3.97055791
##  -2.5944513   0.44216546 -1.05073567 -2.45605853 -0.19547759 -3.44478478
##  -2.06618442 -0.31827406  0.33132461 -1.42034302 -1.99403888 -1.9960294
##  -0.3177321  -0.93105174 -3.33987324 -2.23873728 -2.26682273 -0.97489087
##  -0.11013999 -2.10199721 -4.92044973 -0.2268601   0.17555703 -0.452344
##   0.0997955   0.38170503 -1.12672936 -5.02931752 -0.28382002 -0.40600884
##  -3.54417561 -0.13658187 -2.53492654 -3.59955284  0.24052964 -3.63051952
##  -2.36724255 -2.53407919 -3.15855548  0.29016505 -0.89342143 -0.28440201
##  -1.82705403 -1.97868835 -4.76518696 -3.15199984 -3.64316697 -2.81597459
##   0.2337428   0.23859968 -2.24166875 -3.2230766  -1.00261683  0.26436319
##  -1.00961904 -2.26822203 -1.80392675 -0.45430003]
```

These are the differences from the PyTorch method of calculating MLE's and the closed form expressions.

Problem 3

Suppose that (X,Y) is a random pair where X follows an exponential distribution with parameter $\theta = 1$ and Y takes on values in {0,1}. Suppose that $P[Y = 1|X = x] = \frac{1}{1+x}$ when x>0 or 0 otherwise.

- **Define the Bayes classifier**
- **Calculate its Bayes error**
- **Verify the Bayes error empirically using N=10,000 simulation replicates (you may use any programming language)**

Bayes classifier: We will denote the Bayes classifier as g*.

$$g^*(x) = \begin{cases} 1 \; if \; \frac{1}{1+x} > \frac{1}{2} \\ 0 \; otherwise \end{cases}$$

Bayes error:

$$Note: because \; X \sim Exp(\theta = 1), \; pdf \; can \; be \; written \; as \; exp(-x)$$

$$E(min(\frac{1}{1+x}, 1 - \frac{1}{1+x})) = \int_0^1 1 - \frac{1}{1+x} exp(-x)dx \; + \; \int_1^\infty \frac{1}{1+x} exp(-x)dx$$

$$\approx 0.17 \; + \; 0.13 \; = \; 0.30$$

Simulation study to verify Bayes error:

```
bayeserror <- function(cc, n) { x <- runif(n, 0, 4*cc)
etax <- x/(x+cc)
Y <- rbinom(n, 1, etax)
Yhat <- as.integer(etax > 0.5)
mean(Y!=Yhat) }

bayeserror(.25, 10000)
```

```
## [1] 0.3025
```