Econ 432 Homework 4

Richard Grigorian (UID: 505-088-797)

3/6/23

Contents

1 Part I: Review Questions

1.1 Problem 1

To estimate the risk of a stock, a sample of 73 cc returns was taken and the sample standard devation $\hat{\sigma}$ was 0.057. To get a confidence interval for the true standard deviation σ, 10,000 resamples were taken. Let $\hat{\sigma}_{b,boot}$ were sorted and the table below contains selected values of $\hat{\sigma}_{b,boot}$ ranked from smallest to alrgest (so rank 1 is the smallest and so forth).

Rank	Value of $\hat{\sigma}_{b,boot}$
250	0.047
500	0.049
2500	0.053
7500	0.060
9500	0.065
9750	0.067

Part (a)

Find the bootstrap standard error of $\hat{\sigma}$.

Solution

From the given info, we can infer that we should use the *quantile bootstrap* method for estimating standard error. Recall the expression for this technique is:

$$\text{SE}_{q-bt}(\hat{\theta}) = \frac{\hat{\theta}^*_{[\alpha_1]} - \hat{\theta}^*_{[\alpha_2]}}{z_{\alpha_1} - z_{\alpha_2}}$$

where z_{α_1} and z_{α_2} denote the α_1 and α_2 quantiles of the standard normal distribution. In practiace, we often choose the 75th and 25th percentiles respectively.

Therefore, we apply this expression to the above problem. To find the normal distribution quantiles we use `python`.

```
from scipy.stats import norm

alpha1 = norm.ppf(0.75)
alpha2 = norm.ppf(0.25)
print('alpha_1 =', alpha1)
print('alpha_2 =', alpha2)
```

```
alpha_1 = 0.6744897501960817
alpha_2 = -0.6744897501960817
```

```
se_stdv = (0.06 - 0.053) / (alpha1 - alpha2)
```

This then gives us:

$$\text{SE}_{q-bt}(\hat{\sigma}) = \frac{0.060 - 0.053}{0.674 - (-0.0674)} = 0.005189$$

Part (b)

Find the 90% and 95% confidence intervals using the bootstrap standard error in part (a) and the asymptotic normal distribution of $\hat{\sigma}$.

Solution

The confidence interval expression is:

$$1 - \alpha \,\%\, \text{CI: } \left[\hat{\sigma} - \text{SE}_{q-bt}(\hat{\sigma}) z_{1-\alpha/2} \;,\; \hat{\sigma} + \text{SE}_{q-bt}(\hat{\sigma}) z_{1-\alpha/2}\right]$$

At the 90% level, this would be:

$$90\% \text{ CI: } \left[0.057 \pm 1.645(0.005189)\right] = \left[0.0485 \;,\; 0.0655\right]$$

At the 95% level, this would be:

$$95\% \text{ CI: } \left[0.057 \pm 1.96(0.005189)\right] = \left[0.0468 \;,\; 0.0672\right]$$

Part (c)

Find the 90% and 95% equal tail bootstrap confidence interval for σ.

Solution

To construct an equal tail bootstrap confidence interval we need

$$\left[\hat{\sigma} + c^*_{T,\alpha/2} \;,\; \hat{\sigma} + c^*_{T,1-\alpha/2}\right]$$

where $c^*_{T,\alpha/2}$ is the α qunatile of $\{\hat{\sigma} - \hat{\sigma}^{*,b}\}_{b=1}^{B}$. Conceptually, now the lowest rank value will have the largest difference. Therefore, the ranks for the differences will be somewhat flipped. Therfore, at the 90% level this would be:

$$90\% \text{ Equal Tail CI: } \left[0.057 + (0.057 - 0.065) \;,\; 0.057 + (0.057 - 0.049)\right] = \left[0.049 \;,\; 0.065\right]$$

At the 95% level, this would be:

$$95\% \text{ Equal Tail CI: } [0.057 + (0.057 - 0.067)\ ,\ 0.057 + (0.057 - 0.047)] = [0.047\ ,\ 0.067]$$

1.2 Problem 2

In the following Python program, resampling was used to estimate the bias and variance of the sample correlation between the variables in the vectors x and y.

```
import numpy as np
samplecor = np.corrcoef(x,y)[0,1]
n = len(x)
nboot = 5000
resamplecor = np.zeros(nboot)
for b in range(nboot):
    ind = np.random.choice(np.arrange(0,n,1),
                    replace = True, size = n)
    resamplecor[b] = np.corrcoef(x[ind], y[ind])[0,1]
print(samplecor)
print(np.mean(resamplecor))
print(np.std(resamplecor, ddof = 1))
```

The output is

```
> n
[1] 20
> samplecor
[1] 0.69119
> np.mean(resamplecor)
[1] 0.68431
> np.std(resamplecor, ddof = 1)
[1] 0.11293
```

Part (a)

Estimate the bias of the sample correlation coefficient.

Solution

Recall that bias is simply the difference between our estimate and the true parameter. Hence, the bias of the sample correlation coefficient under bootstrap is:

$$\text{Bias}(\rho, \hat{\rho}) = \mathbb{E}[\hat{\rho}] - \rho = \bar{\hat{\rho}}^* - \hat{\rho} = 0.68431 - 0.69119 = -0.00688$$

Part (b)

Estimate the standard deviation of the sample correlation coefficient.

Solution

From the above printout, see that the standard deviation of our boostrapped sample correlations is equal to 0.11293. Hence, **the standard deviation of the sample correlation coefficient is 0.11293**.

Part (c)

Estimate the MSE of the sample correlation coefficient

Solution

MSE is defined as

$$\text{MSE} = B^{-1} \sum_{b=1}^{B} \left(\hat{\theta}^{*,b} - \hat{\theta}\right)^2$$

With the output above, it would be most convienent to reform MSE using the Bias-Variance decomposition:

$$\text{MSE}(\hat{\theta}^{*,b}, \hat{\theta}) = \left(\text{ Bias}(\hat{\theta}^{*,b}, \hat{\theta})\right)^2 + \text{Var}(\hat{\theta}^{*,b})$$

We have both of these values since variance is just the square of standard deviation. Hence, the MSE of our sample correlation coefficient is:

$$\text{MSE} = (-0.00688)^2 + (0.11293)^2 \approx 0.0128$$

Part (d)

What fraction of the MSE is due to bias? How serious is the bias? Should something be done to reduce the bias? Explain your answer.

Solution

To find the fraction of MSE due to bias, we simply look at our Bias-Variance decomposition once more. It is constructed of Bias and Variance; however, the Bias is squared so we have to account for that when computing the fraction. Therefore, the fraction of MSE due to bias is:

$$\frac{\left(\text{Bias}(\hat{\theta}^{*,b}, \hat{\theta})\right)^2}{\text{MSE}(\hat{\theta}^{*,b}, \hat{\theta})} = \frac{(-0.00688)^2}{0.0128} \approx 0.0037$$

That means that about 0.37% of the MSE is coming from bias. That is not a very large number, in comparison to the variance which accounts for about 99.63% of the MSE. Therefore, while bootstrap bias tends to over-estimate the true moment, it seems to be a fairly minimal issue here. Rather, Variance is the dominant cause of the MSE; hence, we could take steps to reduce the variance at the cost of introducing a bit more bias. That being said, boostrap standard deviation also often over-estimates the true moment, so it is possible that our MSE is actually lower than what we found here.

But in short, the bias is not very serious here and we do not need to reduce it further.

2 Part II: Python Exercises

2.1 Problem 1

This exercise uses weekly data in Apple from the first week of January, 2010 to the last week of January, 2021. Use the adjusted closing prices to compute weekly cc returns. Assume that the returns are i.i.d., even though there may be some autocorrelation and volatility clustering is likely. In bootstrap, let $B = 1000$ and set the random seed to 432 i.e., `np.random.seed(432)` in `Python`.

```
# Initial Imports
import yfinance as yf # pandas_datareader yahoo api not working
import numpy as np
from scipy.stats import iqr, norm

# Import Data
df = yf.download("AAPL",
                 start = "2010-01-01",
                 end = "2021-01-31",
                 interval = '1wk')
df.reset_index(inplace = True)
```

```
[**********100%**********]  1 of 1 completed
```

```
# CC return
Ret = np.log(df['Adj Close']/df['Adj Close'].shift(1))
Ret = np.array(Ret)[1:] # Pass into array
T = len(Ret)
```

Part (a)

Find the sample mean and sample variance of the cc return, and also their standard errors (SEs) using the formula provided in the slides of statistical inference

Solution

```
mu_h = np.mean(Ret) # Sample mean
sig2_h = np.var(Ret, ddof = 1)  # Sample variance
sd_mu = np.sqrt(sig2_h/T) # Standard error of mu_h
sd_sig2 = sig2_h * np.sqrt(2/T) # Standard error of sig2_h

# Print and round to six decimals
print("The sample mean is %.6f" % mu_h)
print("The sample variance is %.6f" % sig2_h)
print("The standard error of mu_hat is %.6f" % sd_mu)
print("The standard error of sig2_hat is %.6f" % sd_sig2)
```

```
The sample mean is 0.005212
The sample variance is 0.001375
The standard error of mu_hat is 0.001542
The standard error of sig2_hat is 0.000081
```

Part (b)

Find the bootsrap SEs (the bootstrap SEs and the bootstrap IQR SEs) of the sample mean and the sample variance. Compare the bootstrap SEs with the SEs in part (a). Are they very different from each other for the same estimator (i.e., the sample mean and the sample variance)?

Solution

```
np.random.seed(432) # Set seed
B = 1000 # Number of bootstrap samples
```

```
boot_samples = np.zeros((T,B)) # Initialize
for b in range(B):
    # random sample at each iteration
    boot_samples[:, b] = np.random.choice(
        Ret,
        replace = True,
        size = T
    )
```

```
# Sample metrics for each iteration [Mean]
boot_means = np.mean(boot_samples, axis = 0)
B_SE_mu = np.std(boot_means, ddof = 1)

IQR_SE_mu = iqr(
    boot_means, interpolation = 'linear' # for boudnary values
    )/(norm.ppf(0.75) - norm.ppf(0.25))

print("The bootstrap SE of mu_hat is %.6f" % B_SE_mu)
print("The IQR SE of mu_hat is %.6f" % IQR_SE_mu)
```

```
The bootstrap SE of mu_hat is 0.001556
The IQR SE of mu_hat is 0.001572
```

```
# Sample metrics for each iteration [Variance]
boot_vars = np.var(boot_samples, ddof = 1, axis = 0)
B_SE_sig2 = np.std(boot_vars, ddof = 1)
IQR_SE_sig2 = iqr(
    boot_vars, interpolation = 'linear'
    )/(norm.ppf(0.75) - norm.ppf(0.25))

print("The bootstrap SE of sig2_hat is %.6f" % B_SE_sig2)
print("The IQR SE of sig2_hat is %.6f" % IQR_SE_sig2)
```

```
The bootstrap SE of sig2_hat is 0.000114
The IQR SE of sig2_hat is 0.000109
```

For the standard error of the sample mean, we see that the bootstrap estimation provides a slightly larger estimate (which is to be expected). And for the standard error of the sample variance, we see similar results with the bootstrap technique delivering a larger standard error. In both instances, the two techniques are different. Although, they are not *very* different as they are similar values.

Part (c)

Find the bootstrap SEs (the bootstrap SEs and the bootstrap IQR SEs) of the sample standard deviation.

Solution

This follows a very similar process to part (b).

```
# SE of St. Dev.
boot_sds = np.std(boot_samples, ddof = 1, axis = 0)
B_SE_sig = np.std(boot_sds, ddof = 1)
IQR_SE_sig = iqr(
    boot_sds, interpolation = 'linear'
    )/(norm.ppf(0.75) - norm.ppf(0.25))

print("The bootstrap SE of sig_hat is %.6f" % B_SE_sig)
print("The IQR SE of sig_hat is %.6f" % IQR_SE_sig)
```

```
The bootstrap SE of sig_hat is 0.001539
The IQR SE of sig_hat is 0.001471
```

Part (d)

Estimate the bias and the mean square error of the sample standard deviation using bootstrap.

Solution

Here we do not need to use the Bias-Variance Decomposition since we have all of the data. Rather, we can just use our bootstrap standard deviation and our sample standard deviation.

```
sig_h = np.std(Ret, ddof = 1) # Sample St. Dev
B_Bias_sig = np.mean(boot_sds) - sig_h # Bias Calculation
B_MSE_sig = np.mean((boot_sds - sig_h) ** 2) # MSE

print("The bootstrap bias of sig_hat is %.6f" % B_Bias_sig)
print("The bootstrap MSE of sig_hat is %.6f" % B_MSE_sig)
```

```
The bootstrap bias of sig_hat is -0.000140
The bootstrap MSE of sig_hat is 0.000002
```

Part (e)

Construct the equal tail and the symmetric 95% bootstrap confidence intervals of the standard deviation of the cc return.

Solution

```
# Quantiles of the difference (refer to formula)
q_et_1 = np.quantile(sig_h - boot_sds, 0.025)
q_et_2 = np.quantile(sig_h - boot_sds, 0.975)
q_sym = np.quantile(np.abs(boot_sds - sig_h ), 0.95)

print("95% Equal Tail CI: ", [sig_h + q_et_1, sig_h + q_et_2])
print("95% Symmetric CI: ", [sig_h - q_sym, sig_h + q_sym])
```

```
95% Equal Tail CI:  [0.03416604242617804, 0.04017247446202637]
95% Symmetric CI:  [0.03409684899894767, 0.04005717279480623]
```

Part (f)

Suppose we invest $1000 on Apple for one week. Find the parametric and nonparametric estimators of VaR(0.1) and ES(0.1) for this investment. for the parametric estimator, you shall assume that returns are normally distributed.

Solution

```
W0 = 1000 # Initial wealth
q1 = norm.ppf(0.1, loc = mu_h, scale = sig_h) # Parametric (Norm)
q2 = np.quantile(Ret, 0.1, method = 'linear') # Nonparametric
```

```
VaR_1 = W0 * (np.exp(q1) - 1)
VaR_2 = W0 * (np.exp(q2) - 1)

print("The parametric estimate of VaR(0.1) is %.6f" % VaR_1)
print("The nonparametric estimate of VaR(0.1) is %.6f" % VaR_2)
```

```
The parametric estimate of VaR(0.1) is -41.421690
The nonparametric estimate of VaR(0.1) is -38.195415
```

As expected, the nonparametric estimate of the VaR is more conservative.

```
# Parametric ES
nsim = 500000
r_B = norm.rvs(
    loc = mu_h,
    scale = sig_h,
    size = nsim,
    random_state = 432)
L_B = W0 * (np.exp(r_B) - 1)
I_B = (L_B <= VaR_1) * 1
ES_1 = np.mean(L_B * I_B)/np.mean(I_B)
```

```
# Nonparametric ES
L_1 = W0 * (np.exp(Ret) - 1)
I_1 = (L_1 <= VaR_2)
ES_2 = np.mean(L_1 * I_1)/np.mean(I_1)

print("The parametric estimate of ES(0.1) is %.6f" % ES_1)
print("The nonparametric estimate of ES(0.1) is %.6f" % ES_2)
```

```
The parametric estimate of ES(0.1) is -57.901513
The nonparametric estimate of ES(0.1) is -62.535302
```

Part (g)

Find the bootstrap SEs (the bootstrap SEs and the bootstrap IQR SEs) of the nonparametric estimators of VaR(0.1) and ES(0.1) in part (f).

Solution

```
# VaR function
def VaR(y, p = 0.1, W0 = W0):
    mu = np.mean(y)
    q = np.quantile(y, p, method = 'linear')
    return W0 * (np.exp(q) - 1)
```

```
# Bootstrap Function
def boot(data, func, nboot=5000, n=T):
```

```
    """
    func: function, VaR or ES
    nboot: number of bootstrap samples
    n: size of a boostrap sample
    """
    result = np.zeros(nboot)
    for i in range(nboot):
        boot_sample = np.random.choice(data, replace = True, size = n)
        result[i] = func(boot_sample)
    return result
```

```
np.random.seed(432) # Set seed

VaR_est = VaR(Ret) # VaR of cc returns

# Bootstrap VaR & SE
boot_VaR = boot(data = Ret, func = VaR)
BootSE_VaR = np.std(boot_VaR, ddof = 1)

# Bootstrap IQR SE
Boot_IQR_SE_VaR = (np.quantile(boot_VaR, 0.75) - np.quantile(boot_VaR, 0.25)
                   )/(norm.ppf(0.75) - norm.ppf(0.25))

print("Bootstrap SE (nonparametric estimator) VaR(0.1): %.6f" % BootSE_VaR)
print("Bootstrap IQR SE (nonparametric estimator) VaR(0.1): %.6f" % Boot_IQR_SE_VaR)
```

```
Bootstrap SE (nonparametric estimator) of VaR(0.1): 2.580610
Bootstrap IQR SE (nonparametric estimator) of VaR(0.1): 2.531376
```

```
# ES function
def ES(y, p=0.1, W0 = W0):
    mu = np.mean(y)
    q = np.quantile(y, p, method = 'linear')
    VaR_np = W0 * (np.exp(q) - 1)
    L_1 = W0 * (np.exp(y) - 1)
    I_1 = (L_1 <= VaR_np) * 1
    return np.mean(L_1 * I_1)/np.mean(I_1)
```

```
np.random.seed(432) # Set seed
```

```
ES_est = ES(Ret) # ES of cc returns

# Bootstrap ES & SE
boot_ES = boot(data = Ret, func = ES)
BootSE_ES = np.std(boot_ES, ddof = 1)

# Bootstrap IQR SE
Boot_IQR_SE_ES = (np.quantile(boot_ES, 0.75) - np.quantile(boot_ES, 0.25)
                  )/(norm.ppf(0.75) - norm.ppf(0.25))

print("Bootstrap SE (nonparametric estimator) ES(0.1): %.6f" % BootSE_ES)
print("Bootstrap IQR SE (nonparametric estimator) ES(0.1): %.6f" % Boot_IQR_SE_ES)
```

```
Bootstrap SE (nonparametric estimator) ES(0.1): 4.400157
Bootstrap IQR SE (nonparametric estimator) ES(0.1): 4.430407
```

Part (h)

Find the 95% confidence intervals of VaR(0.1) and ES(0.1) using the nonparametric method in part (f).

Solution

```
# Bootstrap VaR Confidence Intervals
## Using Boot-SE
Boot_SE_CI095_VaR = [VaR_est - BootSE_VaR*1.96,
                     VaR_est + BootSE_VaR*1.96]
## Using IQR Boot-SE
Boot_IQR_SE_CI095_VaR = [VaR_est - Boot_IQR_SE_VaR * 1.96,
                         VaR_est + Boot_IQR_SE_VaR * 1.96]

print("95% CI using the bootstrap SE: ", Boot_SE_CI095_VaR)
print("95% CI using the bootstrap IQR SE: ", Boot_IQR_SE_CI095_VaR)
```

```
95% CI using the bootstrap SE:  [-43.25341066534851, -33.13741976636861]
95% CI using the bootstrap IQR SE:  [-43.15691123929683, -33.23391919242029]
```

```
# Bootstrap ES Confidence Intervals
## Using Boot-SE
Boot_SE_CI095_ES = [ES_est - BootSE_ES*1.96,
                    ES_est + BootSE_ES*1.96]
## Using IQR Boot-SE
Boot_IQR_SE_CI095_ES = [ES_est - Boot_IQR_SE_ES * 1.96,
                        ES_est + Boot_IQR_SE_ES * 1.96]

print("95% CI using the bootstrap SE: ", Boot_SE_CI095_ES)
print("95% CI using the bootstrap IQR SE: ", Boot_IQR_SE_CI095_ES)
```

```
95% CI using the bootstrap SE:  [-71.15960858113608, -53.91099478925781]
95% CI using the bootstrap IQR SE:  [-71.21889846150403, -53.85170490888987]
```