Applied Statistical Methods

Bootstrap

Xuemao Zhang
East Stroudsburg University

April 14, 2023

Outline

- Introduction
- Bootstrap concept
- Bootstrap method
- Python: Bootstrap
- Python: Block bootstrap

Introduction

- In the section we discuss another resampling method: bootstrap.
- The bootstrap is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- It can provide an estimate of the standard error of a coeffcient, or a confdence interval for that coeffcient.
  - For example, we used bootstrap to estimate a population mean (Lecture 15).

An example

- Suppose that we wish to invest a fxed sum of money in two financial assets that yield returns of X and Y, respectively, where X and Y are random quantities.
- We will invest a fraction α of our money in X, and will invest the remaining $1-\alpha$ in Y.
- We wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $Var[\alpha X + (1-\alpha)Y]$.
- One can show that the value that minimizes the risk is given by

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}},$$

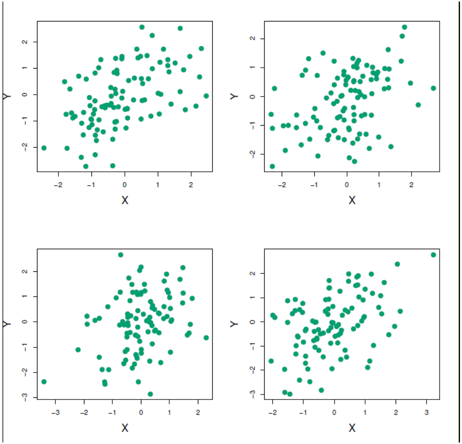
where $\sigma_X^2 = Var(X), \sigma_Y^2 = Var(Y)$, and $\sigma_{XY} = Cov(X, Y)$.

An example

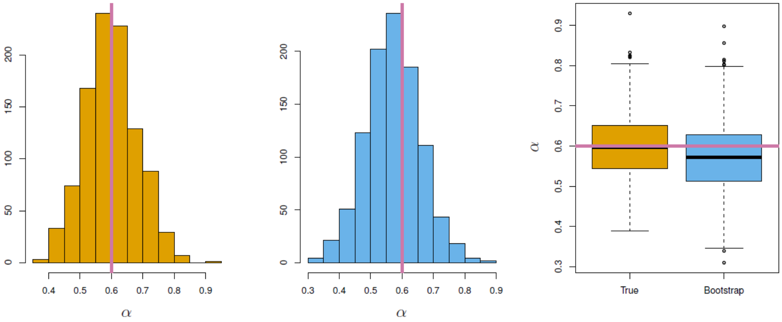
- But the values of σ_X^2, σ_Y^2 and σ_{XY} are unknown.
- We can compute estimates for these quantities, $\hat{\sigma}_X^2, \hat{\sigma}_Y^2$ and $\hat{\sigma}_{XY}$,using a data set that contains measurements for X and Y.
- We can then estimate the value of α that minimizes the variance of our investment using
$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}},$$
- We can estimate α using a single set of data. How do we estimate the standard deviation of $\hat{\alpha}$.

An example

- Each panel displays 100 simulated returns for investments X and Y. From left to right and top to bottom, the resulting estimates for α are 0.576, 0.532, 0.657, and 0.651.



An example



0
50
100
150
200
0.4
0.5
0.6
0.7
0.8
0.9
α
0
50
100
150
200
0.3
0.4
0.5
0.6
0.7
0.8
0.9
α
0.3
0.4
0.5
0.6
0.7
0.8
0.9
α
True
Bootstrap

An example

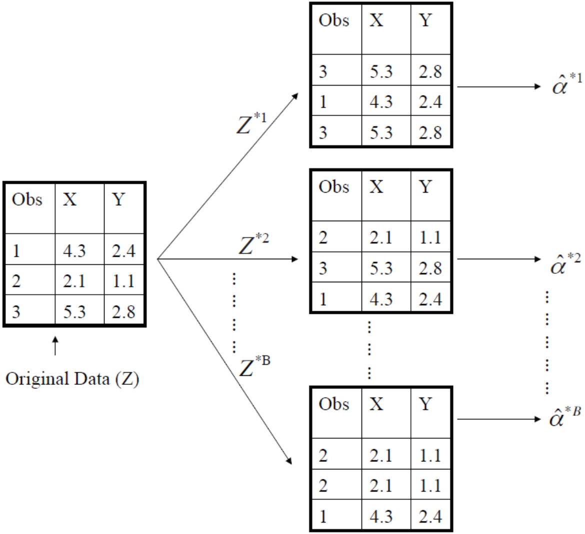
- To estimate the standard deviation of $\hat{\alpha}$, we repeated the process of simulating 1,000 paired observations of X and Y, and estimating α 1,000 times.
- We thereby obtained 1,000 estimates for α, which we can call $\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{1000}$.
- The left-hand panel of the Figure displays a histogram of the resulting estimates.
- For these simulations the parameters were set to $\sigma_X^2 = 1, \sigma_Y^2 = 1.25$ and $\sigma_{XY} = 0.5$, and so we know that the true value of α is 0.6 (indicated by the red line).
- The standard deviation of the 1000 estimates gives us $SE(\hat{\alpha}) \approx 0.0083$

Bootstrap method

- In practice, however, the procedure for estimating $SE(\hat{\alpha})$ outlined above cannot be applied, because for real data we cannot generate new samples from the original population.
- However, the bootstrap approach allows us to use a computer to mimic the process of obtaining new data sets, so that we can estimate the variability of our estimate without generating additional samples.
- Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations from the original data set **with replacement**.
- Each of these "bootstrap data sets" is created by sampling **with replacement**, and is the same size as our original dataset. As a result some observations may appear morethan once in a given bootstrap data set and some not at all.

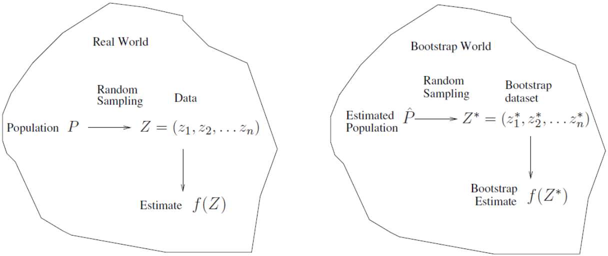
Bootstrap method

- Each bootstrap data set contains n observations, sampled with replacement from the original data set.



Bootstrap method

- A general picture for the bootstrap



- Primarily used to obtain standard errors of an estimate.
- A Bootstrap Percentile confidence interval can be obtained by the distribution of the bootstrap estimates.

Bootstrap method

- In more complex data situations, figuring out the appropriate way to generate bootstrap samples can require some thought.
- For example, if the data is a time series, we can't simply sample the observations with replacement (why not?).
- We can instead create blocks of consecutive observations, and sample those with replacements. Then we paste together sampled blocks to obtain a bootstrap dataset.

Bootstrap method

Can the bootstrap estimate prediction error?

- To estimate prediction error using the bootstrap, we could think about using each bootstrap dataset as our training sample, and the original sample as our validation sample.
- But each bootstrap sample has significant overlap with the original data. About two-thirds of the original data points appear in each bootstrap sample
- This will cause the bootstrap to seriously underestimate the true prediction error.
- The other way around - with original sample = training sample, bootstrap dataset = validation sample - is worse!
- Can partly fix this problem by only using predictions for those observations that did not (by chance) occur in the current bootstrap sample.
- But the method gets complicated, and in the end, cross-validation provides a simpler, more attractive approach for estimating prediction error.

Python: Bootstrap

- Performing a bootstrap analysis in Python entails only two steps.
  - First, we must create a function that computes the statistic of interest.
  - Second, we use the `bootstrap()` function, which is from `scipy.stats`, to perform the bootstrap by repeatedly sampling observations from the data set with replacement.
- We illustrate the use of the bootstrap in the simple example on the `Portfolio` data set.
- We use bootstrap to estimate the accuracy of the linear regression model on the `Auto` data set.

Python: Bootstrap

- Consider the `Portfolio` data set in the `ISLR` package in R.

```
import pandas as pd
Portfolio = pd.read_csv('../data/Portfolio.csv')
Portfolio.head()
```

```
##            X         Y
## 0 -0.895251 -0.234924
## 1 -1.562454 -0.885176
## 2 -0.417090  0.271888
## 3  1.044356 -0.734198
## 4 -0.315568  0.841983
```

Python: Bootstrap

- We first create a function, `alpha_fn()`, which takes as input the (X, Y) data as well as a vector indicating which observations should be used to estimate α.

```
import numpy as np
def alpha_fn(X, Y):
  return ((np.var(Y)-np.cov(X,Y)[0,1])/
(np.var(X)+np.var(Y)-2*np.cov(X,Y)[0,1]))
```

- This function returns an estimate for α from the solution we just saw. For instance, the following command tells python to estimate α using all 100 observations.

```
x=Portfolio.X
y=Portfolio.Y
print(alpha_fn(x,y))
```

```
## 0.5766511516664772
```

Python: Bootstrap

- Next, We use the `sample()` function to randomly select 100 observations from the range 1 to 100, with replacement.
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html

```
import random
random.seed(1)
dfsample = Portfolio.sample(frac=1, replace=True)
x=dfsample.X
y=dfsample.Y
print(alpha_fn(x,y))
```

```
## 0.6414204016475976
```

Python: Bootstrap

- We can implement a bootstrap analysis by performing this command many times, recording all of the corresponding estimates for α, and computing the resulting standard deviation.

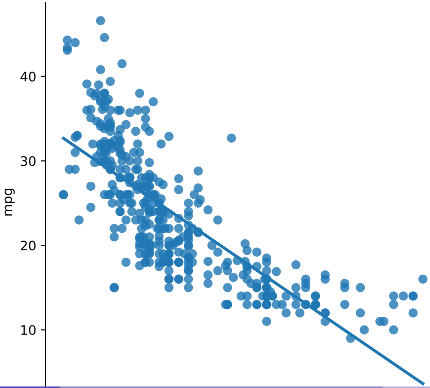
```
def bstrap(df, R):
  import numpy as np
  estimate_boot=np.zeros(R)
  for i in range(0,R):
      dfsample = df.sample(frac=1, replace=True)
      x = dfsample.X
      y = dfsample.Y
      estimate_boot[i] = alpha_fn(x,y)
  return(np.mean(estimate_boot), np.std(estimate_boot))

bstrap(Portfolio, 1000)
```

```
## (0.573447645233924, 0.0912081001073443)
```

Python: Bootstrap in Regression

- We use the bootstrap approach to assess the variability of the estimates of the regression coefficents.
- Consider the `Auto` data set and use `horsepower` to predict `mpg`.

```
import pandas as pd
Auto= pd.read_csv('../data/Auto.csv')
import matplotlib.pyplot as plt
import seaborn as sns
sns.lmplot(data=Auto, x='horsepower', y='mpg',ci=None, fit_reg=True)
```



Python: Bootstrap in Regression

- We first create a simple function, `boot_fn()`, which takes in the `Auto` data set and the number of samples, and returns the standard errors of the bootstrap estimates for the intercept and slope terms.

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
def boot_fn(df, R):
  intercept_boot=np.zeros(R)
  slope_boot=np.zeros(R)
  for i in range(0,R):
      dfsample = df.sample(frac=1, replace=True)
      model=smf.ols(formula='mpg~horsepower', data=dfsample)
      fit=model.fit()
      intercept_boot[i]=fit.params[0]
      slope_boot[i]=fit.params[1]
  return( np.mean(intercept_boot), np.std(intercept_boot),
np.mean(slope_boot), np.std(slope_boot) )
```

```
boot_fn(Auto, 1000)
```

```
## (39.994132348560456, 0.8398902170755723, -0.1583891463241258, 0.00733471725202153
```

Python: Bootstrap in Regression

- Compare the bootstrap estimates with the general linear regression model fit.

```
model=smf.ols(formula='mpg~horsepower', data=Auto)
fit=model.fit()
print(fit.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                    mpg   R-squared:                       0.606
## Model:                            OLS   Adj. R-squared:                  0.605
## Method:                 Least Squares   F-statistic:                     599.7
## Date:                Thu, 13 Apr 2023   Prob (F-statistic):           7.03e-81
## Time:                        14:25:16   Log-Likelihood:                -1178.7
## No. Observations:                 392   AIC:                             2361.
## Df Residuals:                     390   BIC:                             2369.
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     39.9359      0.717     55.660      0.000      38.525      41.347
## horsepower    -0.1578      0.006    -24.489      0.000      -0.171      -0.145
## ==============================================================================
## Omnibus:                       16.432   Durbin-Watson:                   0.920
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.305
## Skew:                           0.492   Prob(JB):                     0.000175
```

Python: Bootstrap in Regression

- We also can consider the quadratic regression model

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
def boot_fn2(df, R):
  beta0_boot=np.zeros(R)
  beta1_boot=np.zeros(R)
  beta2_boot=np.zeros(R)
  for i in range(0,R):
      dfsample = df.sample(frac=1, replace=True)
      model=smf.ols(formula='mpg~horsepower+I(horsepower^2)', data=dfsample)
      fit=model.fit()
      beta0_boot[i]=fit.params[0]
      beta1_boot[i]=fit.params[1]
      beta2_boot[i]=fit.params[2]
  from tabulate import tabulate
  col_names = ["", "Estimate", "Std. Error"]
  output = [["Intercept", np.mean(beta0_boot), np.std(beta0_boot)],
        ["horsepower", np.mean(beta1_boot), np.std(beta1_boot)],
        ["I(horsepower^2)", np.mean(beta2_boot), np.std(beta2_boot)]]
  print(tabulate(output, headers=col_names)  )
```

```
boot_fn2(Auto, 1000)
```

```
##                    Estimate    Std. Error
## ---------------  ----------  ------------
## Intercept         39.9642        0.874706
## horsepower        -0.136747      0.125158
## I(horsepower^2)   -0.021405      0.124721
```

Python: Bootstrap in Regression

```
model2=smf.ols(formula='mpg~horsepower+I(horsepower^2)', data=Auto)
fit2=model2.fit()
print(fit2.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                    mpg   R-squared:                       0.606
## Model:                            OLS   Adj. R-squared:                  0.604
## Method:                 Least Squares   F-statistic:                     299.1
## Date:                Thu, 13 Apr 2023   Prob (F-statistic):           2.12e-79
## Time:                        14:25:24   Log-Likelihood:                -1178.6
## No. Observations:                 392   AIC:                             2363.
## Df Residuals:                     389   BIC:                             2375.
## Df Model:                           2
## Covariance Type:            nonrobust
## ===============================================================================
##                           coef    std err          t      P>|t|      [0.025      0.
## -------------------------------------------------------------------------------
## Intercept              39.9433      0.719     55.534      0.000      38.529      41
## horsepower             -0.1321      0.124     -1.065      0.288      -0.376       (
## I(horsepower ^ 2)      -0.0259      0.124     -0.208      0.835      -0.270       (
## ==============================================================================
## Omnibus:                       16.878   Durbin-Watson:                   0.921
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.833
## Skew:                           0.499   Prob(JB):                     0.000134
## Kurtosis:                       3.307   Cond. No.                         456.
```

Python: Block bootstrap

- Suppose there is a time series data set.

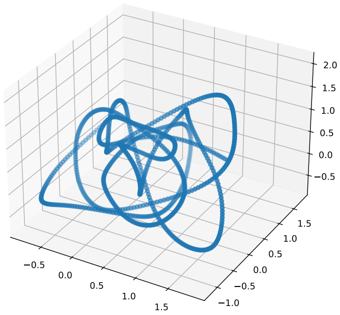
```
Xy=pd.read_csv("../data/Xy.csv")
Xy.head(10)
```

```
##      t        X1        X2         y
## 0    1  1.297720  0.805921  0.298968
## 1    2  1.267323  0.799034  0.318134
## 2    3  1.236882  0.792169  0.337201
## 3    4  1.206317  0.785296  0.356121
## 4    5  1.175553  0.778385  0.374842
## 5    6  1.144513  0.771404  0.393312
## 6    7  1.113118  0.764324  0.411482
## 7    8  1.081305  0.757109  0.429304
## 8    9  1.049059  0.749703  0.446740
## 9   10  1.016378  0.742045  0.463757
```

Python: Block bootstrap

- 3-d plot: https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(Xy['X1'], Xy['X2'], Xy['y'])
plt.show()
```



Python: Block bootstrap

```python
import statsmodels.api as sm
import statsmodels.formula.api as smf
model3=smf.ols(formula='y~X1+X2', data=Xy)
fit3=model3.fit()
print(fit3.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                      y   R-squared:                       0.117
## Model:                            OLS   Adj. R-squared:                  0.115
## Method:                 Least Squares   F-statistic:                     66.14
## Date:                Thu, 13 Apr 2023   Prob (F-statistic):           1.06e-27
## Time:                        14:25:30   Log-Likelihood:                -810.66
## No. Observations:                1000   AIC:                             1627.
## Df Residuals:                     997   BIC:                             1642.
## Df Model:                           2
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept      0.2658      0.020     13.372      0.000       0.227       0.305
## X1             0.1453      0.026      5.604      0.000       0.094       0.196
## X2             0.3134      0.029     10.722      0.000       0.256       0.371
## ==============================================================================
## Omnibus:                        9.703   Durbin-Watson:                   0.002
## Prob(Omnibus):                  0.008   Jarque-Bera (JB):               13.694
```

Python: Block bootstrap

- First consider the general bootstrap method for the simple linear regression model.

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
from tabulate import tabulate
def boot_fn3(df, R):
  beta0_boot=np.zeros(R)
  beta1_boot=np.zeros(R)
  beta2_boot=np.zeros(R)
  for i in range(0,R):
      dfsample = df.sample(frac=1, replace=True)
      model=smf.ols(formula='y~X1+X2', data=dfsample)
      fit=model.fit()
      beta0_boot[i]=fit.params[0]
      beta1_boot[i]=fit.params[1]
      beta2_boot[i]=fit.params[2]
  col_names = ["", "Estimate", "Std. Error"]
  output = [["Intercept", np.mean(beta0_boot), np.std(beta0_boot)],
        ["X1", np.mean(beta1_boot), np.std(beta1_boot)],
        ["X2", np.mean(beta2_boot), np.std(beta2_boot)]]
  print(tabulate(output, headers=col_names)  )
```

Python: Block bootstrap

```
boot_fn3(Xy, 1000)
```

```
##              Estimate    Std. Error
## ---------  ----------  ------------
## Intercept    0.266028     0.0148548
## X1           0.144487     0.0284277
## X2           0.311909     0.0354403
```

Python: Block bootstrap

- Finally, use the **block bootstrap** to estimate $s.e.(\hat{\beta}_1)$. Use blocks of 100 contiguous observations, and resample ten whole blocks with replacement then paste them together to construct each bootstrap time series. For example, one of bootstrap resamples could be:

```
#Concatenating the ranges
new_rows=[*range(100,200), *range(400,500), *range(100,200),
*range(900,1000),*range(300,400), *range(0,100), *range(0,100),
*range(800,900),*range(200,300), *range(700,800)]
new_Xy=Xy.iloc[new_rows, ]
new_Xy.shape
```

```
## (1000, 4)
```

```
new_Xy.head()
```

```
##         t        X1        X2         y
## 100   101  0.327028  0.054913  1.135742
## 101   102  0.358675  0.053752  1.134353
## 102   103  0.389793  0.053699  1.132062
## 103   104  0.420223  0.054714  1.128780
## 104   105  0.449808  0.056759  1.124419
```

Python: Block bootstrap

- Now let's write the new bootstrap function.

```
import numpy as np
from random import choices
from tabulate import tabulate

def newboot_fn(df, R):
  beta0_boot=np.zeros(R)
  beta1_boot=np.zeros(R)
  beta2_boot=np.zeros(R)

  for i in range(0,R):
    seq=choices(list(range(10)), k=10)
    newrows=[]
    for j in range(10):
      row_boot=choices( range(seq[j]*100, (seq[j]+1)*100), k=100) #numpy array of length 100
      newrows.extend(row_boot)

    new_df=df.iloc[newrows, ] #a block bootstrap sample

    model=smf.ols(formula='y~X1+X2', data=new_df)
    fit=model.fit()
    beta0_boot[i]=fit.params[0]
    beta1_boot[i]=fit.params[1]
    beta2_boot[i]=fit.params[2]

  col_names = ["", "Estimate", "Std. Error"]
  output = [["Intercept", np.mean(beta0_boot), np.std(beta0_boot)],
  ["X1", np.mean(beta1_boot), np.std(beta1_boot)],
  ["X2", np.mean(beta2_boot), np.std(beta2_boot)]]
  print(tabulate(output, headers=col_names)  )
```

Python: Block bootstrap

```
newboot_fn(Xy, 1000)
```

```
##                 Estimate    Std. Error
## ---------  ----------  ------------
## Intercept    0.278887     0.0919399
## X1           0.137507     0.201131
## X2           0.393714     0.324099
```

License