# Lecture7
# 重抽样方法

刘晨辉
邮箱：chenhuiliu@hnu.edu.cn
办公室：土木楼A422

2023.04.11

# 目录

# 1. 验证集法(Validation Set Approach)

我们建立模型后，总是需要对模型表现进行评价(Model Assessment)。

- 训练错误(Training Error)：所建统计模型对训练数据的预测错误
- 检验错误(Test Error)：所建统计模型对检验数据的预测错误

如何评价?

最好用新的数据集来对模型进行检验，查看test error rate，但是很多时候没有新的数据集。那该怎么办?

在没有很大的检验数据集的情况下，我们需要留下(Hold out)观测数据中的一部分，作为验证数据(Validation data)。

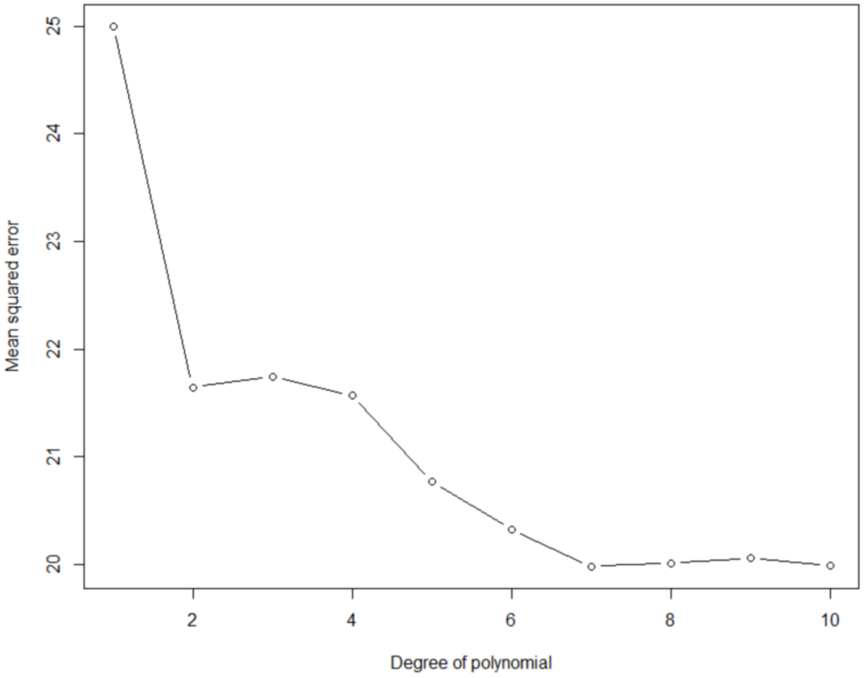
# 1. 验证集法(Validation Set Approach)

针对Auto数据集，求出不同多项式对应的均方误差。

```
```{r libraries}
library(boot)
library(doBy)
library(ggplot2)
library(ISLR2)
library(MASS)
```
```

```
## 1.1 Validation set
# 抽取一半数据为train数据
train_no<- sample(x = nrow(Auto),size = nrow(Auto)/2)
print(train_no)
# 产生train数据
Auto_train<- Auto[train_no,]
# 产生test数据
Auto_test<- Auto[-train_no,]
for(i in 1:10){
  # poly(): 多项式函数
  glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data = Auto)
  Auto_test$pred<- predict(glm1,Auto_test)
  # 计算MSE(mean squared error): 均方误差
  mse<- mean((Auto_test$mpg-Auto_test$pred)^2)
  if(i==1){
    mses<- mse
  }else{
    mses<- c(mses,mse)
  }
}
mses<- data.frame(mses)
mses$dof<- c(1:10)

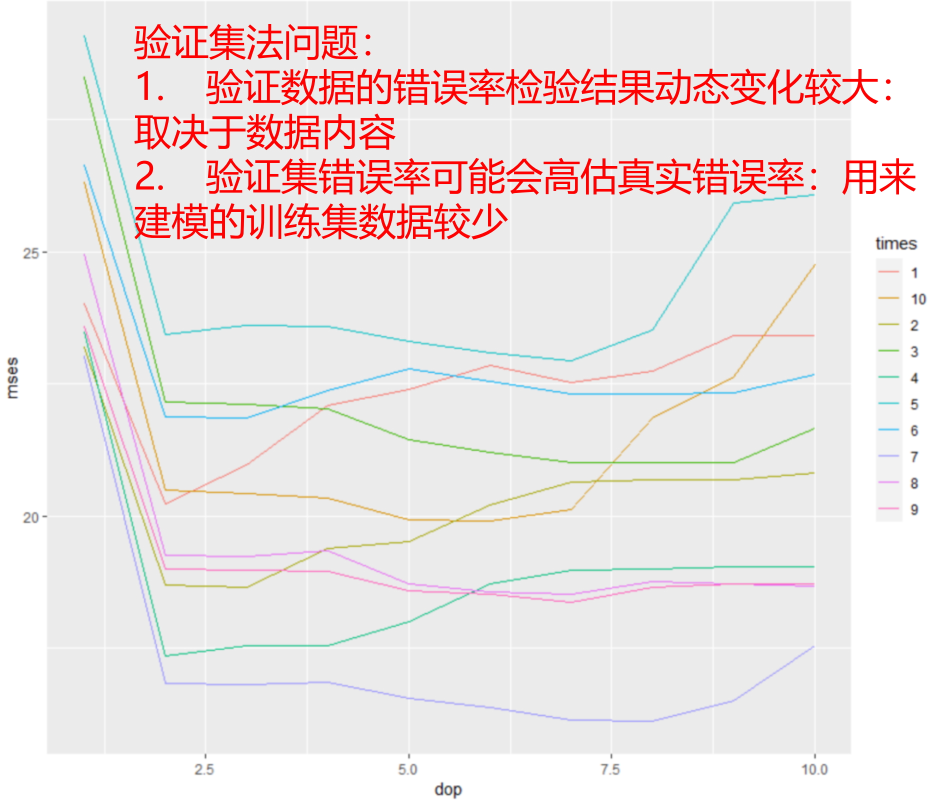
# 画出多项式次数与MSE的关系
plot1<- plot(mses$dof,mses$mses,type = "b",
          xlab="Degree of polynomial",ylab = "Mean squared error")
print(plot1)
```



# 1. 验证集法(Validation Set Approach)

针对Auto数据集，求出不同多项式对应的均方误差。

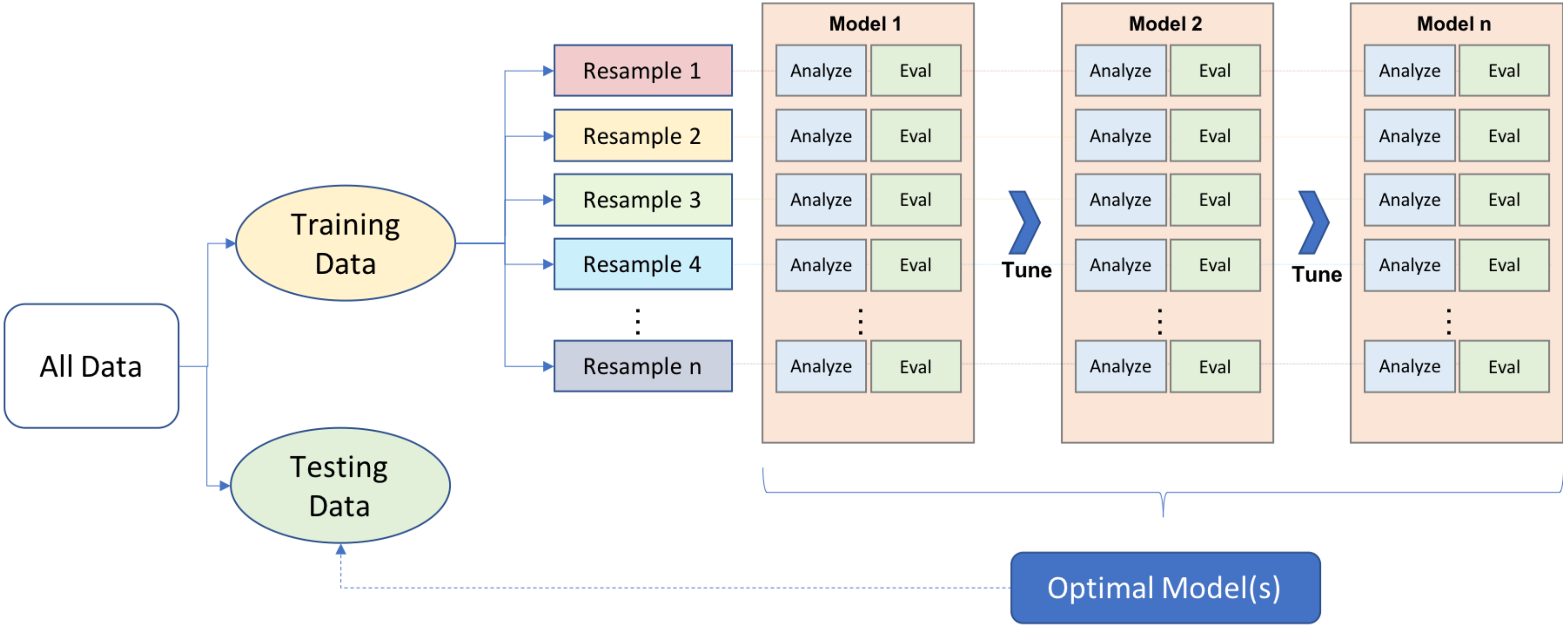
```
## 1.2 Multiple degrees + Multiple samples
for(k in 1:10){
  train_no<- sample(nrow(Auto),nrow(Auto)/2)
  Auto_train<- Auto[train_no,]
  Auto_test<- Auto[-train_no,]
  for(i in 1:10){
    glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data =Auto_train)
    Auto_test$pred<- predict(glm1,Auto_test)
    mse<- mean((Auto_test$mpg-Auto_test$pred)^2)
    if(i==1){
      mses<- mse
      }else{
        mses<- c(mses,mse)
      }
    }
  mses<- data.frame(mses)
  mses$dop<- c(1:10)
  mses$times<- k
  if(k==1){
    msess<- mses
  }else{
    msess<- rbind(msess,mses)
  }
}
msess$times<- as.factor(as.character(msess$times))
# plot the data
g1<- ggplot(data=msess)+
  geom_line(aes(x=dop,y=mses,col=times))
print(g1)
```



# 1. 验证集法(Validation Set Approach)

- 如何评价模型表现？是选取表现好的，还是选择表现差的？

我们既不选择最好的，也不选择最差的，而是重复进行多次，分析这些指标的平均值。



# 2. 交叉抽样(Cross-Validation)

## 2.1 留一法交叉验证(Leave-One-Out CV, LOOCV)

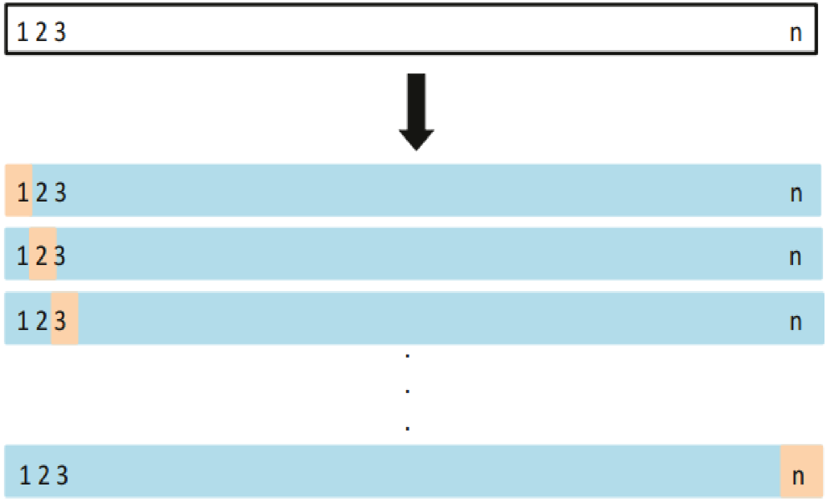
假设数据集含有n个数据$\{(x_1,y_1),\ldots,(x_n,y_n)\}$，LOOCV每次选择1个数据作为验证集，采用MSE(Mean Squared Error)作为评价指标

（1）$(x_1,y_1)$作为验证集，其余$(n-1)$个数据作为训练集$\{(x_2,y_2),\ldots,(x_n,y_n)\}$: $MSE_1=(y_1-\hat{y}_1)^2$

（2）$(x_2,y_2)$作为验证集，其余$(n-1)$个数据作为训练集$\{(x_1,y_1),\ldots,(x_n,y_n)\}$: $MSE_2=(y_2-\hat{y}_2)^2$

…

（n）$(x_n,y_n)$作为验证集，其余$(n-1)$个数据作为训练集$\{(x_1,y_1),\ldots,(x_{n-1},y_{n-1})\}$: $MSE_n=(y_n-\hat{y}_n)^2$




LOOCV评价指标是$n$个检测误差的平均值

$$CV_{(n)}=\frac{1}{n}\sum_{i=1}^{n}MSE_i$$

# 2. 交叉抽样(Cross-Validation)

## 2.1 留一法交叉验证(Leave-One-Out CV, LOOCV)

### LOOCV手动计算

```
library(doBy)
## 2.1 LOOCV: 手动
for(k in 1:nrow(Auto)){
  test_no<- k
  Auto_train<- Auto[-test_no,]
  Auto_test<- Auto[test_no,]
  for(i in 1:10){
    glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data = Auto_train)
    Auto_test$pred<- predict(glm1,Auto_test)
    mse<- mean((Auto_test$mpg-Auto_test$pred)^2)
    if(i==1){
      mses<- mse
      }else{
        mses<- c(mses,mse)
      }
    }
  mses<- data.frame(mses)
  mses$dop<- c(1:10)
  mses$times<- k
  if(k==1){
    msess<- mses
  }else{
    msess<- rbind(msess,mses)
  }
}
msess$times<- as.factor(as.character(msess$times))
# 查看结果
loocv_mse<- summaryBy(mses~dop,data=msess);print(loocv_mse)
names(loocv_mse)<- c("dop","mses_manual")
print(loocv_mse)
```

```
> print(loocv_mse)
   dop mses_manual
1    1    24.23151
2    2    19.24821
3    3    19.33498
4    4    19.42443
5    5    19.03321
6    6    18.97864
7    7    18.83305
8    8    18.96115
9    9    19.06863
10  10    19.49093
```

### LOOCV工具包计算

```
## 2.2 LOOCV: 工具包
library(boot)
# 对于1次方公式，求LOOCV的test error
glm1<- glm(mpg~horsepower,data = Auto)
loocv_mse1<- cv.glm(Auto,glm1)
print(loocv_mse1)

# 对于多项式公式，求LOOCV的test error
loocv_mse_boot<- rep(0,10)
for(i in 1:10){
    glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data = Auto)
    loocv_mse_boot[i]<- cv.glm(Auto,glm1)$delta[1]
}
loocv_mse$mses_boot<- loocv_mse_boot
print(loocv_mse)
```

```
> loocv_mse1
$call
cv.glm(data = Auto, glmfit = glm1)

$K
[1] 392

$delta
[1] 24.23151 24.23114
```

```
> print(loocv_mse)
   dop mses_manual mses_boot
1    1    24.23151  24.23151
2    2    19.24821  19.24821
3    3    19.33498  19.33498
4    4    19.42443  19.42443
5    5    19.03321  19.03321
6    6    18.97864  18.97864
7    7    18.83305  18.83305
8    8    18.96115  18.96115
9    9    19.06863  19.06863
10  10    19.49093  19.49093
```

# 2. 交叉抽样(Cross-Validation)

## 2.1 留一法交叉验证(Leave-One-Out CV, LOOCV)

LOOCV优点：

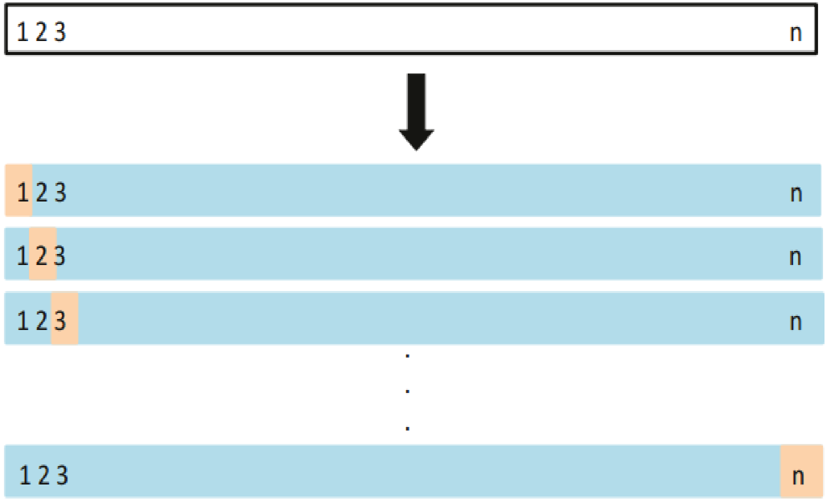
（1）偏差更小：

使用的训练数据集有$(n-1)$个数据，接近整个数据集，所以不大可能会高估检验错误率(Test error rate)。

（2）结果稳定：

验证集法每次的估计结果都可能发生变化，因为训练和验证集是随机分割的。对于LOOCV，每次运行产生的结果是不变的。

（3）可以与任何预测模型结合使用。




LOOCV缺点：

计算量巨大，需要计算$n$次。

# 2. 交叉抽样(Cross-Validation)

## 2.2 K-重交叉验证(K-Fold CV)

将数据集平均分成k组(k-fold)，K-重交叉验证每次选择1组数据作为验证集，采用MSE(Mean Squared Error)作为评价指标：

（1）第1组作为验证集，其余$(k-1)$组数据作为训练集，计算$MSE_1$。
（2）第2组作为验证集，其余$(k-1)$组数据作为训练集，计算$MSE_2$。
…
（n）第k组作为验证集，其余$(k-1)$组数据作为训练集，计算$MSE_k$。

k-fold CV评价指标是$k$个检测误差的平均值

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

可以发现：

- k-重交叉验证可以大大减少计算量，但也不会对模型结果造成太大影响。
- LOOCV是k-fold CV的一个特殊情况
- 一般选择$k=5$，或者$k=10$

# 2. 交叉抽样(Cross-Validation)

## 2.2 K-重交叉验证(K-Fold CV)

```
### 3. k-fold CV
# 采集现在的时间
t0<- Sys.time()
# 10-fold
set.seed(17)
kcv_mse_10<- rep(1,10)
for(i in 1:10){
    glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data = Auto)
    # 注意现在要加上K=10
    kcv_mse_10[i]<- cv.glm(Auto,glm1,K=10)$delta[1]
}
t1<- Sys.time()
print(kcv_mse_10)
```

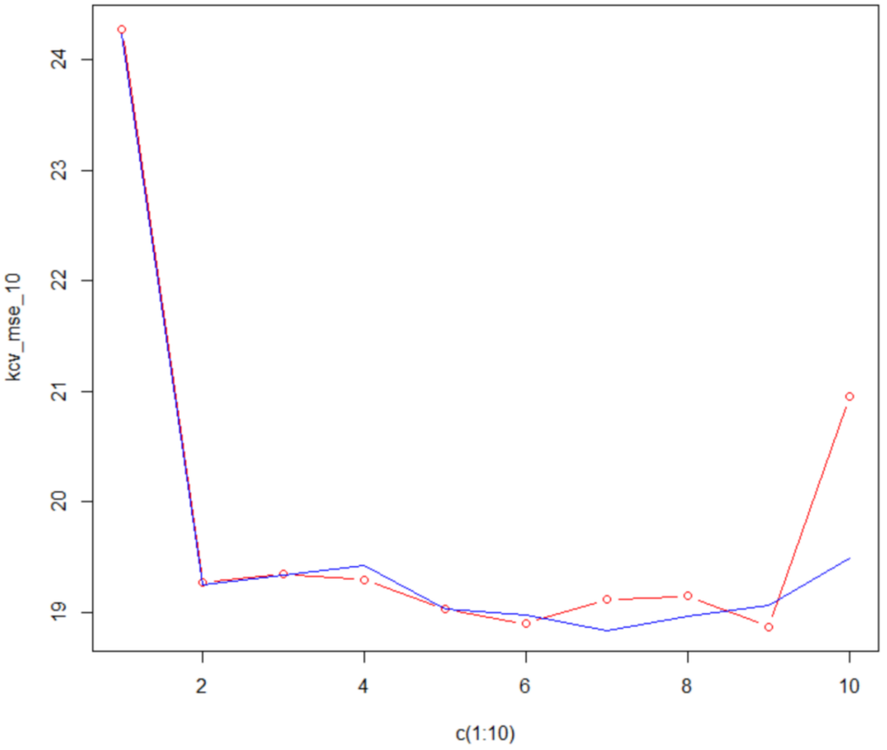
```
> print(kcv_mse_10)
 [1] 24.27207 19.26909 19.34805 19.29496 19.03198
 [6] 18.89781 19.12061 19.14666 18.87013 20.95520
```

```
# LOOCV
kcv_mse_n<- rep(1,10)
for(i in 1:10){
    glm1<- glm(mpg~poly(horsepower,i,raw = TRUE),data = Auto)
    # 注意现在要加上K=10
    kcv_mse_n[i]<- cv.glm(Auto,glm1)$delta[1]
}
t2<- Sys.time()

# 检查两个模型的结果
plot2<- plot(x=c(1:10),y=kcv_mse_10,col="red",type = "b")+
  lines(x=c(1:10),y=kcv_mse_n,col="blue")

# 检查两个模型的估算时间
print(t1-t0)
print(t2-t1)
```

```
> # 检查两个模型的估算时间
> print(t1-t0)
Time difference of 0.2709961 secs
> print(t2-t1)
Time difference of 9.570046 secs
```




10-fold CV与LOOCV相比没有明显区别，但运行时间大大缩短！

# 2. 交叉抽样(Cross-Validation)

## 2.3 分类数据交叉验证

对于LOOCV方法，当数据为分类数据时(Classification)，采用错误率为评价指标：

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i$$

$$Err_i = I(y_i \neq \hat{y}_i)$$

对于k-重交叉验证，

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_k$$

$$Err_k = \frac{1}{n_k}\sum_{j\in n_k} I(y_j \neq \hat{y}_j)$$

# 3. 自抽样(Bootstrap)

假设我们有一笔钱，可以投向两个项目，每个项目的回报分别为$X$和$Y$，两者均为随机数值。我们打算将一部分钱$\alpha$，$0 < \alpha < 1$，投入回报为$X$的项目；剩下的钱$1-\alpha$，投入回报为$Y$的项目。那么，我们投资的预计回报为$\alpha X + (1-\alpha)Y$。、

由于每个项目的投资均有波动性(Variability)，我们希望能够选择一个$\alpha$来最小化整体投资风险，也就是方差(Variance)。换句话说，我们想要最小化$Var(\alpha X + (1-\alpha)Y)$。

如何达到这个目的?

$$\begin{aligned} Var(\alpha X + (1-\alpha)Y) &= Var(\alpha X) + Var\big((1-\alpha)Y\big) + 2Cov(\alpha X, (1-\alpha)Y) \\ &= \alpha^2 Var(X) + (1-\alpha)^2 Var(Y) + 2\alpha(1-\alpha)Cov(X,Y) \\ &= \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\alpha(1-\alpha)\sigma_{XY} \end{aligned}$$

式中：$\sigma_X^2 = Var(X)$，$\sigma_Y^2 = Var(Y)$，$\sigma_{XY} = Cov(X,Y) = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{n}$。

求导得：$\frac{\partial(Var(\alpha X+(1-\alpha)Y))}{\partial\alpha} = 2\alpha\sigma_X^2 + 2(1-\alpha)(-1)\sigma_Y^2 + 2(1-\alpha)\sigma_{XY} - 2\alpha\sigma_{XY} = 0$

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

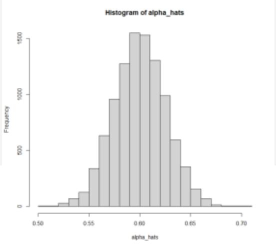
# 3. 自抽样(Bootstrap)

假设$\sigma_X^2=1$， $\sigma_Y^2=1.25$，$\sigma_{XY}=0.5$。我们利用仿真数据集查看一下结果。

```
library(MASS)
############ 设置Bivariate normal分布参数 ##########
sample_size<- 1000
mean1<- 0; mean2<- 0
mean_vector<- c(mean1,mean2)
var1<- 1; var2<- 1.25
cov12<- 0.5
covariance_matrix<- matrix(c(var1,cov12,cov12,var2),nrow = 2)
alpha<- (var2-cov12)/(var1+var2-2*cov12)
print(alpha) # 真实的alpha为0.6

########### 仿真1次：每次产生1000个样本数据 ###############
# 利用mvrnorm产生1000个数据
set.seed(123)
sample_dat<- mvrnorm(n = sample_size,
                     mu = mean_vector,
                     Sigma =covariance_matrix)
# 估计bivariate normal分布的各参数值
var1_hat<- var(sample_dat[,1])
var2_hat<- var(sample_dat[,2])
cov12_hat<- cov(sample_dat[,1],sample_dat[,2])
# 估计alpha
alpha_hat<- (var2_hat-cov12_hat)/(var1_hat+var2_hat-2*cov12_hat)
print(alpha_hat)
```

```
########### 仿真10000次 ###############
for(i in 1:10000){
  # 仿真1000个样本数据
  sample_dat<- mvrnorm(n = sample_size,
                       mu = mean_vector,
                       Sigma =covariance_matrix)
  # 估计bivariate normal分布的各参数值
  var1_hat<- var(sample_dat[,1])
  var2_hat<- var(sample_dat[,2])
  cov12_hat<- cov(sample_dat[,1],sample_dat[,2])
  # 估计alpha
  alpha_hat<- (var2_hat-cov12_hat)/(var1_hat+var2_hat-2*cov12_hat)
  if(i==1){
    alpha_hats<- alpha_hat
  }else{
    alpha_hats<- c(alpha_hats,alpha_hat)
  }
}
print(c(mean(alpha_hats),var(alpha_hats)))
print(hist(alpha_hats))
```




```
> print(c(mean(alpha_hats),var(alpha_hats)))
[1] 0.6002164051 0.0006437424
```

# 3. 自抽样(Bootstrap)

在实际情况中，我们很难会有那么多样本数据集。Bootstrap提供了一个新的途径，来模仿产生新样本数据的过程。与上述在总体数据中抽取不同的样本数据集不同，Bootstrap是通过对1个原始的样本数据集进行可放回的重复抽样。

Bootstrap方法：假设我们有一个含有$n$个数据的数据集，我们目标是得到某统计值$a$。

（1）对数据进行可放回抽样，得到含有$n$个数据的新数据集，计算目标统计值$\hat{\alpha}_1$；

（2）对数据进行可放回抽样，得到含有$n$个数据的新数据集，计算目标统计值$\hat{\alpha}_2$；

…

（k）对数据进行可放回抽样，得到含有$n$个数据的新数据集，计算目标统计值$\hat{\alpha}_k$；

$$\hat{a} = \frac{1}{k}\sum_{i=1}^{k}\hat{\alpha}_i$$

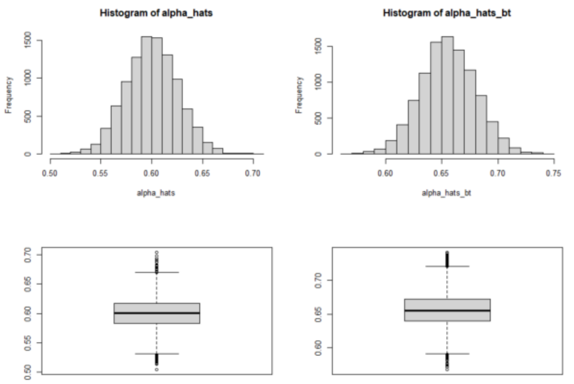
$$Var(\hat{a}) = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(\hat{\alpha}_i - \hat{a})^2}$$

# 3. 自抽样(Bootstrap)

Bootstrap是通过对1个原始的样本数据集进行可放回的重复抽样：手动

```r
########### Bootstrap: 每次产生1000个样本数据 ###############
## 手动Bootstrap
set.seed(123)
sample_dat1<- mvrnorm(n = sample_size,
                      mu = mean_vector,
                      Sigma =covariance_matrix)
for(i in 1:10000){
  # 仿真10000个样本数据
  sample_no<- sample(nrow(sample_dat1),nrow(sample_dat1),replace = TRUE)
  sample_dat<- sample_dat1[sample_no,]
  # 估计bivariate normal分布的各参数值
  var1_hat<- var(sample_dat[,1])
  var2_hat<- var(sample_dat[,2])
  cov12_hat<- cov(sample_dat[,1],sample_dat[,2])
  # 估计alpha
  alpha_hat<- (var2_hat-cov12_hat)/(var1_hat+var2_hat-2*cov12_hat)
  if(i==1){
    alpha_hats_bt<- alpha_hat
  }else{
    alpha_hats_bt<- c(alpha_hats_bt,alpha_hat)
  }
}
```

```r
# 对比两个数据集alpha估计值：平均值与标准差
print(c(mean(alpha_hats),sqrt(var(alpha_hats))))
print(c(mean(alpha_hats_bt),sqrt(var(alpha_hats_bt))))
# 把视图图改成1*2分布
par(mfrow=c(2,2))
p1<- hist(alpha_hats)
p2<- hist(alpha_hats_bt)
p3<- boxplot(alpha_hats)
p4<- boxplot(alpha_hats_bt)
par(mfrow=c(1,1))
```



# 3. 自抽样(Bootstrap)

Bootstrap是通过对1个原始的样本数据集进行可放回的重复抽样：使用工具包

```r
###############
### 工具包Bootstrap
sample_dat1<- mvrnorm(n = sample_size,
                      mu = mean_vector,
                      Sigma =covariance_matrix)
sample_no<- sample(nrow(sample_dat1),nrow(sample_dat1),replace = TRUE)
# 新建一个函数
alpha_fun<- function(sample_dat1,sample_no){
  sample_dat<- sample_dat1[sample_no,]
  # 估计bivariate normal分布的各参数值
  var1_hat<- var(sample_dat[,1])
  var2_hat<- var(sample_dat[,2])
  cov12_hat<- cov(sample_dat[,1],sample_dat[,2])
  # 估计alpha
  alpha_hat<- (var2_hat-cov12_hat)/(var1_hat+var2_hat-2*cov12_hat)
  # 返回alpha估计值
  return(alpha_hat)
}
## 检查一下新建函数：
#第一个sample_dat1/sample_no是参数名，函数建成之后不可变
#第二个sample_dat1/sample_no是参数输入值，可变，与函数无关
print(alpha_fun(sample_dat1 = sample_dat1,
                sample_no = sample_no))
## Bootstrap抽样数据：boot函数
boot1<- boot(sample_dat1,alpha_fun,R=10000)
print(boot1)
## 对比手动计算结果与boot函数结果
print(c(mean(alpha_hats_bt),sqrt(var(alpha_hats_bt))))
```

MASS工具包：boot

```
> print(boot1)

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = sample_dat1, statistic = alpha_fun, R = 10000)


Bootstrap Statistics :
     original        bias    std. error
t1* 0.6345113 0.0001404943   0.0261419
> ## 对比手动计算结果与boot函数结果
> print(c(mean(alpha_hats_bt),sqrt(var(alpha_hats_bt))))
[1] 0.65566713 0.02435523
```

手动计算结果与工具包计算结果会非常接近！

# 3. 自抽样(Bootstrap)

例子1：利用Auto数据集，分析线性模型回归参数的方差。

```
########### Bootstrap例子1: 回归系数估计 ################
## 直接估算参数值
lm1<- lm(mpg~horsepower,data = Auto)
print(coef(lm1))

## Bootstrap估算
# 新建参数估计函数: coef_fun
coef_fun<- function(data,index){
  lm1<- lm(mpg~horsepower,data = data[index,])
  coef(lm1)
}
# 检查一下新建函数
print(coef_fun(data = Auto,index = 1:nrow(Auto)))
print(coef_fun(Auto,sample(392,392,replace = T)))
print(coef_fun(Auto,sample(392,392,replace = T)))

# 利用Boot函数
boot2<- boot(Auto,coef_fun,1000)
print(boot2)
# 对比直接估算结果
print(summary(lm1))
```

```
> print(boot2)

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = coef_fun, R = 1000)


Bootstrap Statistics :
      original        bias    std. error
t1* 39.9358610  0.0281678425 0.856896511
t2* -0.1578447 -0.0002127381 0.007429671

> print(summary(lm1))

Call:
lm(formula = mpg ~ horsepower, data = Auto)

Residuals:
     Min       1Q   Median       3Q      Max
-13.5710  -3.2592  -0.3435   2.7630  16.9240

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 39.935861   0.717499   55.66   <2e-16 ***
horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
```

两者并不完全一致

# 3. 自抽样(Bootstrap)

最常用的一种是.632 Boostrap，假设给定的数据集包含n个样本。该数据集有放回地抽样n次，产生n个样本的训练集。这样原数据样本中的某些样本很可能在该样本集中出现多次。没有进入该训练集的样本最终形成检验集（测试集）。显然每个样本被选中的概率是1/n，因此未被选中的概率就是(1-1/n)，这样一个样本在训练集中没出现的概率就是n次都未被选中的概率，即$(1-1/n)^n$。当n趋于无穷大时，这一概率就将趋近于$e^{-1}=0.368$，所以留在训练集中的样本大概就占原来数据集的63.2%。

$$p=\left(1-\frac{1}{n}\right)^{n}$$

$$\frac{1}{p}=\left(1-\frac{1}{n}\right)^{-n}=\left(\frac{n-1}{n}\right)^{-n+1-1}=\left(\frac{n-1}{n}\right)^{-n+1}*\left(\frac{n-1}{n}\right)^{-1}=\left(\frac{n}{n-1}\right)^{n-1}*\left(\frac{n}{n-1}\right)$$

$$=\left(\frac{n-1+1}{n-1}\right)^{n-1}*\left(\frac{n}{n-1}\right)=\left(1+\frac{1}{n-1}\right)^{n-1}*\left(\frac{n}{n-1}\right)$$

因为$\left(1+\frac{1}{x}\right)^{x}=e$，当x趋向于正无穷大。所以当$n$趋向正无穷大，$\frac{1}{p}=e$

$$p=e^{-1}=0.368$$

# 谢 谢！