Bootstrapping and Permutation

1 Bootstrap

Bootstrapping is any test or metric that uses random sampling with replacement (e.g. mimicking the sampling process), and falls under the broader class of resampling methods (like jackknife, Permutation tests, cross validation, etc.). The bootstrap was published by Bradley Efron in "Bootstrap methods: another look at the jackknife" (1979), inspired by earlier work on the jackknife. Subsequently, he wrote several papers on this, exploring many directions and improving its performance. The term 'bootstrap' is due to the idiom 'pull oneself up by one's bootstraps'.

One of the most important application of bootstrapping is to compute the standard error or standard deviation of 'any' statistical estimate. Unless we can apply central limit theorem (CLT), we cannot easily quantify its standard deviation. Instead of relying on CLT, bootstrapping attempts to learn the distribution of a parameter approximately and then use the approximate distribution to derive its standard deviation.

Now, how can we learn an approximate distribution? Here is an example.

If $P(\theta)$ does not follow any known parametric distribution family. We then approximately learn the distribution $P(\theta)$ using samples $\theta_1, \ldots, \theta_K$ such that $\theta_i \sim P(\theta)$. To see that this is indeed valid. If θ follows a Beta distribution with parameters 19 and 133 ($\theta \sim \text{Beta}(19, 133)$), we want to learn the distribution of $\log(\theta)$.

```
theta <- rbeta(10000, 19, 133)

#Density of log(\theta) using Jacobian is given below
#densijacobian <- (1/beta(19,133)) * exp(log(theta))^19 * (1-exp(log(theta)))^(133-1)
#Use that to compute density values in following grid

thetagrid <- (1:1000)/1000 #seq(range(theta)[1], range(theta)[2], length.out = 1000)
denjacobian <- (1/beta(19,133)) * exp(log(thetagrid))^19 * (1-exp(log(thetagrid)))^(133-1)

plot(density(log(theta)), col=1, type = 'l') #ploting density using Monte Carlo method,
                                             #just transformed the generated data in the
                                             #first line and computed numeric desity
points(log(thetagrid), denjacobian, col=2, type = 'l') #Jacobian computed densities

#Some standard distributions:
```

```
plot(density(theta), col=1, type = 'l') #ploting density using Monte Carlo method
                                        #(computed numeric density)
points(thetagrid, dbeta(thetagrid, 19, 133), col=2, type = 'l')#r function computed density


x <- rnorm(1000, 0, 1)
xgrid <- seq(-3,3,length.out = 1000)

plot(density(x), col=1, type = 'l') #ploting density using Monte Carlo method
                                    #(computed numeric density)
points(xgrid, dnorm(xgrid, 0, 1), col=2, type = 'l') #r function computed density
```

Above examples show numerically computed densities from samples are matching with exact densities.

Hence, if we have samples of the parameter of interest, we can learn its distribution. We can imagine a dataset as a sample of an unknown data generating mechanism. Ok, that makes a dataset one single sample. But we need multiple samples (or datasets). 'Resampling' then comes in action. We will resample from the same dataset multiple times to create these fake data.

If $x_1, \ldots, x_n \sim F$, some distribution, let $\mu = \mathbb{E}(x)$ be our parameter of interest. An unbiased estimate of μ in most cases is sample mean $\hat{\mu} = \frac{1}{n}\sum_i x_i = \bar{x}$. Now, variance of this estimate $V(\hat{\mu}) = V(\frac{1}{n}\sum_i x_i) = \frac{\sigma^2}{n}$, where $V(x_i) = \sigma^2$. When σ^2 is also unknown, we can plug-in the unbiased sample estimate $\hat{\sigma}^2 = \frac{1}{n-1}\sum_i (x_i - \bar{x})^2$.

For mean, standard error or variance estimation is easy. However, for sample median or mode, this is not so easy. Even when we want a variance of $\hat{\sigma}^2$, it is still complicated. More generally, say, T_n be a sample statistic which is a function of the data $\{x_1, \ldots, x_n\}$. Thus $T_n = g(x_1, \ldots, x_n)$.

Monte Carlo method: We can draw multiple samples from the same data generating mechanism (this is also called replication!). For each sample, we can compute the T_n and thus we get a sample of T_n's.

However, for real data, we only have one sample. Thus, Monte Carlo method will not work. Although, we can empirically estimate the distribution F_n of F using the data $x_1, \ldots, x_n$ and generate multiple replications of the data.

By definition, $F(x) = P(X \leq x)$ is a cumulative distribution function. We can estimate F with the empirical distribution function F_n, the cdf that puts mass $1/n$ at each data

point x_i.

$$F_n(x) = \frac{\text{Number of samples in the data} \leq x}{n}.$$

The above definition of empirical distribution is itself complicated when we are in the multivariate regime, as there is no clear definition of the cumulative distribution function. Hence, resampling-based methods are easier to use.

Parametric bootstrap: If we know the parametric class of F, we can consider the parametric bootstrap. Let us assume that we know $x_1, \ldots, x_n$ follows a Poisson distribution. We first estimate the model parameters and then generate samples from the distribution with the estimated parameter.

```
lambdahat = mean(x)
for b = 1,...,B
sample xb_1,...,xb_n from Poisson(lambdahat).
Compute Tb
end;
barT=mean(Tb)
VarT=V(Tb) #this is sample variance
```

Non-parametric bootstrap: what if the distribution is not known at all? Hence, we need to rely on empirically estimated distribution. Then, how to sample from the empirical distribution? Drawing $\{x_1^*, \ldots, x_n^*\}$ from the empirical distribution F_n is equivalent to draw n observations, with replacement from the original data $\{x_1, \ldots, x_n\}$. Therefore, Bootstrapping sampling is also described as resampling data.

```
for b = 1,...,B
sample xb_1,...,xb_n from {x_1,...,x_n} with replacement.
Compute Tb
end;
barT=mean(Tb)
VarT=V(Tb) #this is sample variance
```

Here Tb could be sample median or mode of the sampled data xb_1,...,xb_n. V(Tb)=$\frac{1}{B-1}\sum_b (Tb - \text{mean}(Tb))^2$.

Bootstrap confidence interval

Bootstrapping can further help to construct confidence intervals. There are two ways to do that using the bootstrap samples. Let $T_1, \ldots, T_B$ be the estimates collected from B

many resamples.

Normal approximation: In this approach, we assume normality of the generated samples $\mathbf{T} = T_1, \ldots, T_B$. And thus the interval will be $(mean(\mathbf{T}) - Z_{1-\alpha/2} sd(\mathbf{T}), mean(\mathbf{T}) + Z_{\alpha/2} sd(\mathbf{T}))$ for $100(1-\alpha)\%$ Confidence interval.

Empirical quantile: We can apply the quantile function on $\mathbf{T}$ to compute the confidence interval empirically. Specifically, it will look like $quantile(\mathbf{T}, probs = c(0.025, 0.975))$.

Connection to log(θ) example: We need to generate samples of T_n which is a function of the data $g(x_1, \ldots, x_n)$ (which is like $\log(\theta)$). There, we generated samples of θ. Here we need to get samples of the data i.e. $\{x_1^{(k)}, \ldots, x_n^{(k)}\}_{k=1}^K$. And our samples of $T_n = \{T_n^{(1)}, \ldots, T_n^{(K)}\}$ are $T_n^{(k)} = g(x_1^{(k)}, \ldots, x_n^{(k)})$.

A numerical illustration that resampling preserves the original distribution in 'Expectation': Here we use the statistical concept that cumulative distribution function $F(m) = P(X \leq m)$ 'uniquely' characterizes a distribution. Since we check this numerical, we consider empirical cumulative distribution function (ecdf) for testing. First the code:

```
n <- 100
x <- rnorm(n)

###Number of bootstrap samples is N
N <- 1000

####Store the bootstrap samples
y <- matrix(0, N, n)
for(i in 1:N){
  y[i,] <- x[sample(1:n, replace = T)]
}

m <- 0.7

##True empirical cumulative distribution
mean(x <= m)

##Average empirical cumulative distributions across the resamples
avgecdf <- apply(y, 1, FUN=function(z){mean(z <= m)})
```

```
mean(avgecdf)

sd(avgecdf)

###Empirical confidence interval
quantile(avgecdf, probs = c(0.025,0.975))



####################For standard deviation as estimator############
sd(x)

##Average empirical cumulative distributions across the resamples
avgsd <- apply(y, 1, sd)
mean(avgsd)

sd(avgsd)

###Empirical confidence interval
quantile(avgsd, probs = c(0.025,0.975))
```

We can vary N and n to see their influences. We see that the average ecdf from resamples match with the true ecdf. Note that `mean(x <= m)` or mathematically, $F_n(m) = \frac{\text{Number of samples in the data} \leq m}{n}$ is an estimator of $F(X \leq m)$. Hence, by the bootstrap theory, `sd(avgecdf)` quantifies the variance of the estimator $F_n(m)$. If we have sufficiently large N, in general, the mean and sd of the estimator are primarily influenced by n only. However, if one wants to compute the confidence intervals empirically, it is important to set N large enough to be able to approximate the two tail probabilities sufficiently well.

2 Permutation:

Permutation assisted inference methods are routinely used in various avenues. Such inference methods are usually distribution-free and also does not require any asymptotic result. Below is an example of a permutation test for testing equality of mean in two samples.

If we observe $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$, how can we test if the mean of X and mean of Y have a difference. Consider the follow one-sided test:

- $H_0 : \mu_x = \mu_y$
- $H_A : \mu_x < \mu_y$

2.1 Permutation test for equality in mean

```
#Difference in mean does not exist
n <- 100
m <- 90

x <- rnorm(n,0,1)
y <- rnorm(m,0,1)

label <- c(rep(1, n), rep(2, m))

xy <- c(x, y)

dif <- mean(xy[which(label==2)]) - mean(xy[which(label==1)])

B <- 1000

TB <- numeric(B)

for(b in 1:B){
  set.seed(b)

  labelc <- sample(label)
  TB[b] <- mean(xy[which(labelc==2)]) - mean(xy[which(labelc==1)])
}

pvalue = mean(TB >= dif)
pvalue


#Difference in mean does exist
n <- 100
m <- 90
```

```
x <- rnorm(n,0,1)
y <- rnorm(m,0.5,1)

label <- c(rep(1, n), rep(2, m))

xy <- c(x, y)

dif <- mean(xy[which(label==2)]) - mean(xy[which(label==1)])

B <- 1000

TB <- numeric(B)

for(b in 1:B){
  set.seed(b)

  labelc <- sample(label)
  TB[b] <- mean(xy[which(labelc==2)]) - mean(xy[which(labelc==1)])
}

pvalue = mean(TB >= dif)
pvalue
```

Explanation What is p-value? (From Oxford dictionary) the probability that a particular statistical measure, such as the mean or standard deviation, of an assumed probability distribution will be greater than or equal to (or less than or equal to in some instances) observed results.

If we want to compute this probability without any 'assumed probability distribution', we can use permutation test. Null hypothesis generally assumes equality of two group. Thus undernull, permutation of a dataset would not change the joint probability of the sample under exchangeability.

Loosely, exchangeability means $P(x_1, \ldots, x_n) = P(x_{\pi(1)}, \ldots, x_{\pi(n)})$, where $\{\pi(1), \ldots, \pi(n)\}$ is a permutation of $\{1, \ldots, n\}$. If $n = 3$, by exchangeability $P(x_1, x_2, x_3) = P(x_1, x_3, x_2) = P(x_2, x_3, x_1) = \cdots$ etc. In case of iid (independent and identically distributed) assumption, $P(x_1, x_2, x_3) = P(x_1)P(x_2)P(x_3)$. So the requirement for exchangeability holds vacuously.

Technical detail: Permutation tests are non-parametric tests that solely rely on the assumption of exchangeability.

To get a p-value, we randomly sample (without replacement) possible permutations of our variable of interest. The p-value is the proportion of samples that **have a test statistic larger than that of our observed data** (the notion of p-value). Time series data is rarely exchangeable. To conduct permutation test for time-series data, we modify this approach.

In our equality in mean example, the test statistic is the difference in mean of the two population. Then we shuffle the class labels and compute the difference in mean for each shuffled data-copy and compute the proportion of cases where this value is more extreme than the original difference.

Had the alternative been $\mu_X \neq \mu_Y$, what would have been our test statistic?
Nice visual illustration of permutation test: `https://www.jwilber.me/permutationtest/`.

2.2 Permutation importance in a regression setting

Permutation of the data can also be used to quantify importance of a predictor. As given in [1], the steps are given in Figure 1. Examples of s is R^2, *negative-mean squared error* (why negative? as, by definition of s, we require a measure in which a higher number indicates a better model.)

4.2.1. Outline of the permutation importance algorithm

- Inputs: fitted predictive model m, tabular dataset (training or validation) D.
- Compute the reference score s of the model m on data D (for instance the accuracy for a classifier or the R^2 for a regressor).
- For each feature j (column of D):
  - For each repetition k in $1, \ldots, K$:
    - Randomly shuffle column j of dataset D to generate a corrupted version of the data named $\tilde{D}_{k,j}$.
    - Compute the score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$.
  - Compute importance i_j for feature f_j defined as:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j}$$

Figure 1: Steps for evaluating permutation importance. Source: scikit-learn package

In general, Python package `scikit-learn` and R package `caret` have many in-built functions for different models.

```
x <- rnorm(100)
y <- rnorm(100, 2*x)

z <- rnorm(100) #y does not depend on this z.

X <- cbind(x, z)
###############Permutation importance using R^2 as s ########################
fit <- lm(y~X-1)
tab <- summary(fit)
s <- tab$r.squared
smat <- matrix(0, 50, 2)
for(k in 1:50){
  for(j in 1:2){
    ind <- sample(1:100)
    X1 <- X

    X1[, j] <- X1[ind, j]

    fit <- lm(y~X1-1)

    tab <- summary(fit)
    smat[k, j] <- s- tab$r.squared
  }
}

colMeans(smat)


###############Permutation importance using -negative mean-squared error as s ###############
fit <- lm(y~X-1)

s <- -mean(fit$residuals^2)
smat <- matrix(0, 50, 2)
for(k in 1:50){
  for(j in 1:2){
    ind <- sample(1:100)
    X1 <- X
```

```
    X1[, j] <- X1[ind, j]

    fit <- lm(y~X1-1)


    smat[k, j] <- s- (-mean(fit$residuals^2))
  }
}

colMeans(smat)
```

References

[1] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20(177):1–81, 2019.