Assignment 3

Emre Beray Boztepe

06 12 2023

Definitions:

$V_i(t) = \mathbf{1}\{p_i < t\}, F_n(t) = \frac{\sum V_i}{n}$

Higher Criticism Test: $HC^* = max_{1/n<t<1/2}\sqrt{n}\frac{F_n(t)-t}{\sqrt{t(1-t)}}$

Modification by Stepanova and Pavlenko (2014): $HC_{mod} = max_{0<t<1}\sqrt{n}\frac{F_n(t)-t}{\sqrt{t(1-t)q(t)}}$, $q(t) = $

Question 1:

For n ∈ {5000; 50000} estimate the probability of the type I error for HCmod using the asymptotic critical value for 0.05 significance test Ccrit = 4.14. Defined values to be used

```
set.seed(19191)     # Set seed for reproducibility
K = 1000 #defined K as 1000
n_values = c(5000, 50000)
c_crit = 4.14
```

Function for calculating q(t) by its formula

```
calculate_qt = function(t)
{
  operation = 1/(t*(1-t))
  return(log(log(operation)))
}
```

Function for calculating HCmod

```
calculate_hc_mod = function(n, t)
{
  qt_result = calculate_qt(t)
  divider = sqrt(t*(1-t)*qt_result)
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}
```

Generate random values, calculate HCmod for K times. Calculate Probability Type I error by checking if HCmod mean is higher than c_crit value

```
for (n in n_values) {
  cat("(Type I Error | n = ", n,") =", mean(replicate(K, calculate_hc_mod(n,
runif(n)) >= c_crit), na.rm=T), "\n\n")
}
```

```
## (Type I Error | n =  5000 ) = 0.071
##
## (Type I Error | n =  50000 ) = 0.056
```

Comments:

So, according to the result, when n increases, Probability Type I error decreases.

Question 2:

For n = 5000 estimate critical values of both Higher-Criticism tests at the significance level α = 0.05. Defined variables

```
set.seed(19191)     # Set seed for reproducibility
K = 1000
n = 5000
alpha = 0.05
```

Function for calculating q(t)

```
calculate_qt = function(t)
{
  operation = 1/(t*(1-t))
  return(log(log(operation)))
}
```

Function for calculating HCmod

```
calculate_hc_mod = function(t)
{
  qt_result = calculate_qt(t)
  divider = sqrt(t*(1-t)*qt_result)
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}
```

Function for calculating HC

```
calculate_hc = function(t)
{
  divider = sqrt(t*(1-t))
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}
```

Generate random values for uniform dist, calculate hc and hc_mod and calculate quantiles to find critical values

```
calculate_HC_tests = replicate(K, {x = runif(n); c(calculate_hc(x),
calculate_hc_mod(x))})
critical_HC = quantile(calculate_HC_tests[1, ], 1-alpha, na.rm = T)
critical_mHC = quantile(calculate_HC_tests[2, ], 1-alpha, na.rm = T)
```

Printing the calculated critical values for both tests

```
cat("Critical value for HC:", round(critical_HC, 3), "\n")

## Critical value for HC: 4.579

cat("Critical value for HCmod:", round(critical_mHC, 3), "\n")

## Critical value for HCmod: 4.497
```

Comments: Both estimated critical values for both tests are higher than the critical value (Ccrit) which is provided in the previous question as 4.14. It suggests that the test statistics for both tests are beyond the critical threshold. This means that we would reject the null hypothesis.

Question 3:

Let n = 5000 and a. mu_1: one needle of length $1.2\sqrt{2log(n)}$, other mu's are 0

b. mu_2: 100 needles of length $1.02\sqrt{2log\left(\frac{n}{200}\right)}$, other mu's are 0

c. mu_2: 1000 needles of length $1.002\sqrt{2log\left(\frac{n}{2000}\right)}$, other mu's are 0

Use the above settings to compare the power of the following tests and summarize the results: - Higher-Criticism, - modified Higher-Criticism, - Bonferroni, - chi-square, - Fisher, - Kolmogorov-Smirnov (ks.test), - Anderson-Darling (ad.test {goftest}).

Install and load required packages

```
#install.packages(c("goftest", "VGAM"))

library(goftest)
library(VGAM)

## Warning: package 'VGAM' was built under R version 4.3.2

## Loading required package: stats4

## Loading required package: splines

library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.3.2
```

```
set.seed(19191)     # Set seed for reproducibility
```

Function for calculating q(t) for HC and all other functions to calculate expected tests

```
calculate_qt = function(t)
{
  operation = 1/(t*(1-t))
  return(log(log(operation)))
}

# function for calculating HCmod
calculate_hc_mod = function(t)
{
  qt_result = calculate_qt(t)
  divider = sqrt(t*(1-t)*qt_result)
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}

# function for calculating HC
calculate_hc = function(t)
{
  divider = sqrt(t*(1-t))
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}

# function for testing if found HC stats higher than critical HC
HC_test = function(p_vec) {
  return(as.numeric(calculate_hc(p_vec) > critical_HC))
}

# function for testing if found HC stats higher than critical HCmod
mHC_test = function(p_vec) {
  return(as.numeric(calculate_hc_mod(p_vec) > critical_mHC))
}

# function for calculating Bonferroni
Bonferroni_test = function(p_vec) {
  return(as.numeric(min(p_vec) < 0.05 / length(p_vec)))
}

# function for calculating chi-squared
Chi2_test = function(X) {
  return(as.numeric(sum(X ^ 2) > qchisq(0.95, length(X))))
}
```

```r
# function for calculating fisher test
Fisher_test = function(p_vec) {
  return(as.numeric(-2 * sum(log(p_vec)) > qchisq(0.95, 2 * length(p_vec))))
}

# function for calculating Kolmogorov Smirnov test
KS_test = function(p_vec) {
  return(as.numeric(ks.test(p_vec, runif, alternative = "greater")$p.value <
0.05))
}

# function for calculating Anderson-Darling test
AD_test = function(p_vec) {
  return(as.numeric(ad.test(p_vec)$p.value < 0.05))
}
```

Defined variables

```r
n = 5000
alpha = 0.05
K = 1000
```

Defined 3 mu values

```r
mu1 = c(1.2 * sqrt(2 * log(n)), rep(0, n - 1))
mu2 = c(rep(1.02 * sqrt(2 * log(n / 200)), 100), rep(0, n - 100))
mu3 = c(rep(1.002 * sqrt(2 * log(n / 2000)), 1000), rep(0, n - 1000))
```

Generate data and calculate critical values for HC and HCmod to calculate power

```r
calculate_HC_critical_values = replicate(K, {x = runif(n);
c(calculate_hc(x),
  calculate_hc_mod(x))})
critical_HC = quantile(calculate_HC_critical_values[1, ], .95, na.rm = T)
critical_mHC = quantile(calculate_HC_critical_values[2, ], .95, na.rm = T)
```

Function for calculating power for every test

```r
run_all_tests = function(X) {
  p_vec = 1 - pnorm(X)
  return(c(HC_test(p_vec),
           mHC_test(p_vec),
           Bonferroni_test(p_vec),
           Chi2_test(X),
           Fisher_test(p_vec),
           KS_test(p_vec),
           AD_test(p_vec)))
}
```

Running functions. this way, all powers will be calculated and stored in a matrix.

```
test_results = matrix(nrow=7, ncol=3)
i = 1
for (mu in list(mu1, mu2, mu3)) {
  X = replicate(K, rnorm(length(mu), mu))
  run_test = sapply(1:dim(X)[2], function (j) run_all_tests(X[, j]))
  test_results[1:7, i] = rowMeans(run_test)
  i = i+1
}
```

#create and adjust a table for showing all the results.

```
method_names = c("HC_test",
                 "mHC_test",
                 "Bonferroni",
                 "Chi2",
                 "Fisher",
                 "KS",
                 "AD")
rownames(test_results) = method_names
colnames(test_results) = c("mu_1", "mu_2", "mu_3")
kable(test_results)
```

	mu_1	mu_2	mu_3
HC_test	0.773	1.000	1.000
mHC_test	0.726	1.000	1.000
Bonferroni	0.770	0.991	0.835
Chi2	0.078	1.000	1.000
Fisher	0.087	1.000	1.000
KS	1.000	1.000	1.000
AD	0.059	0.996	1.000

Comments:

According to this table, it can be said that KS test is the best performed test by having value 1.0 for each mu's. For mu1, AD test is the worst performed, for mu2 and mu3, Bonferonni is the worst. Only in Bonferroni, when number of signals increases, power of the test decreases.

Question 4:

Consider the sparse mixture model $f(\mu) = (1-\epsilon)\delta_0 + \epsilon\delta_\mu$

with $\epsilon = n^{-\beta}$ and $\mu = \sqrt{2rlogn}$ for each settings $\beta = [0.6,0.8]$, $r = [0.1,0.4]$ and $n = [5000,50000]$

Option A:

Simulate the critical values for the Neyman-Pearson test in the sparse mixture ## Option B: Compare the power of the Neyman-Pearson test to the power of both versions of HC, Bonferroni, Fisher and chi-square.

Summarize the results referring to the theory learned in class. Definitions for loop variables

```
betas = c(0.6, 0.8)
rs = c(0.1, 0.4)
ns = c(5000, 50000)
set.seed(19191)     # Set seed for reproducibility
```

Function for sparse mixture with specific epsilon and mu

```
likelihood_sparse_mixture = function(X, eps, mu) {
  sum(log((1 - eps) + eps * exp(mu * X - mu^2/2)))
}
```

Function for calculating q(t) for HC and all other tests

```
calculate_qt = function(t)
{
  operation = 1/(t*(1-t))
  return(log(log(operation)))
}

# function for calculating HCmod
calculate_hc_mod = function(t)
{
  n = length(t)
  qt_result = calculate_qt(t)
  divider = sqrt(t*(1-t)*qt_result)
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}

# function for calculating HC
calculate_hc = function(t)
{
  n = length(t)
  divider = sqrt(t*(1-t))
  divided = ecdf(t)(t) - t

  return(max(sqrt(n) * (divided/divider)))
}

# function for testing if found HC stats higher than critical HC
HC_test = function(p_vec) {
  return(as.numeric(calculate_hc(p_vec) > critical_HC))
```

```
}

# function for testing if found HC stats higher than critical HCmod
mHC_test = function(p_vec) {
  return(as.numeric(calculate_hc_mod(p_vec) > critical_mHC))
}

# function for calculating Bonferroni
Bonferroni_test = function(p_vec) {
  return(as.numeric(min(p_vec) < 0.05 / length(p_vec)))
}

# function for calculating chi-squared
Chi_squared_test = function(X) {
  return(as.numeric(sum(X ^ 2) > qchisq(0.95, length(X))))
}

# function for calculating fisher test
Fisher_test = function(p_vec) {
  return(as.numeric(-2 * sum(log(p_vec)) > qchisq(0.95, 2 * length(p_vec))))
}
```

Generate data and calculate critical values for HC and HCmod to calculate power

```
calculate_HC_critical_values = replicate(10000, {x = runif(5000);
c(calculate_hc(x), calculate_hc_mod(x))})
critical_HC = quantile(calculate_HC_critical_values[1, ], .95, na.rm = T)
critical_mHC = quantile(calculate_HC_critical_values[2, ], .95, na.rm = T)
```

Function for performing all test and this function returns results in a vector

```
all_tests = function(eps, mu, C) {
  X = c(rnorm(eps * n, mu), rnorm((1 - eps) * n))
  p_vec = 1 - pnorm(X)
  return(c(likelihood_sparse_mixture(X, eps, mu) >= C,
           HC_test(p_vec),
           mHC_test(p_vec),
           Bonferroni_test(p_vec),
           Chi_squared_test(X),
           Fisher_test(p_vec)))
}
```

Array for storing results for each beta, r and test

```
results = array(dim=c(length(betas),
                      length(rs),
                      length(ns),
                      6))
```

Inside this loop, critical value for the NP test is simulated and power for all tests are calculated (option a and b together)

```
for (b_ in seq(length(betas))) {
  for (r_ in seq(length(rs))) {
    for (n_ in seq(length(ns))) {
      b = betas[b_]; r = rs[r_]; n = ns [n_]

      eps = n ^ (-b)
      mu = sqrt(2 * r * log(n))

      Ts = replicate(1000, likelihood_sparse_mixture(rnorm(n), eps, mu))
      C = quantile(Ts, 0.95)

      # Option A
      print(sprintf("Critical value of Neyman-Person for n = %s, b = %s, r =
%s: %s",
                    n, b, r, round(C, 3)))

      # Option B
      res_tmp = replicate(1000, all_tests(eps, mu, C))
      results[b_, r_, n_, ] = rowMeans(res_tmp, na.rm=T)
    }
  }
}
```

```
## [1] "Critical value of Neyman-Person for n = 5000, b = 0.6, r = 0.1:
1.031"
## [1] "Critical value of Neyman-Person for n = 50000, b = 0.6, r = 0.1:
1.11"
## [1] "Critical value of Neyman-Person for n = 5000, b = 0.6, r = 0.4: -
1.441"
## [1] "Critical value of Neyman-Person for n = 50000, b = 0.6, r = 0.4: -
7.289"
## [1] "Critical value of Neyman-Person for n = 5000, b = 0.8, r = 0.1:
0.263"
## [1] "Critical value of Neyman-Person for n = 50000, b = 0.8, r = 0.1:
0.176"
## [1] "Critical value of Neyman-Person for n = 5000, b = 0.8, r = 0.4:
1.003"
## [1] "Critical value of Neyman-Person for n = 50000, b = 0.8, r = 0.4:
1.226"
```

```
method_names = c("Neyman_Person",
                 "HC_test",
                 "HC_mod_test",
                 "Bonferroni",
                 "Chi_squared",
                 "Fisher")
```

Create a table using values that we have calculated

```
dimnames(results) = list(b=betas, r=rs, n=ns, method = method_names)
results_df = as.data.frame(ftable(results))
```

```
kable(results_df, digits = 3)
```

b	r	n	method	Freq
0.6	0.1	5000	Neyman_Person	0.218
0.8	0.1	5000	Neyman_Person	0.071
0.6	0.4	5000	Neyman_Person	0.993
0.8	0.4	5000	Neyman_Person	0.304
0.6	0.1	50000	Neyman_Person	0.218
0.8	0.1	50000	Neyman_Person	0.055
0.6	0.4	50000	Neyman_Person	1.000
0.8	0.4	50000	Neyman_Person	0.309
0.6	0.1	5000	HC_test	0.112
0.8	0.1	5000	HC_test	0.064
0.6	0.4	5000	HC_test	0.959
0.8	0.4	5000	HC_test	0.259
0.6	0.1	50000	HC_test	0.091
0.8	0.1	50000	HC_test	0.052
0.6	0.4	50000	HC_test	1.000
0.8	0.4	50000	HC_test	0.321
0.6	0.1	5000	HC_mod_test	0.108
0.8	0.1	5000	HC_mod_test	0.055
0.6	0.4	5000	HC_mod_test	0.879
0.8	0.4	5000	HC_mod_test	0.190
0.6	0.1	50000	HC_mod_test	0.103
0.8	0.1	50000	HC_mod_test	0.045
0.6	0.4	50000	HC_mod_test	0.988
0.8	0.4	50000	HC_mod_test	0.236
0.6	0.1	5000	Bonferroni	0.098
0.8	0.1	5000	Bonferroni	0.062
0.6	0.4	5000	Bonferroni	0.815
0.8	0.4	5000	Bonferroni	0.240
0.6	0.1	50000	Bonferroni	0.085
0.8	0.1	50000	Bonferroni	0.049
0.6	0.4	50000	Bonferroni	0.935
0.8	0.4	50000	Bonferroni	0.284
0.6	0.1	5000	Chi_squared	0.128

b	r	n	method	Freq
0.8	0.1	5000	Chi_squared	0.054
0.6	0.4	5000	Chi_squared	0.655
0.8	0.4	5000	Chi_squared	0.108
0.6	0.1	50000	Chi_squared	0.131
0.8	0.1	50000	Chi_squared	0.056
0.6	0.4	50000	Chi_squared	0.649
0.8	0.4	50000	Chi_squared	0.067
0.6	0.1	5000	Fisher	0.187
0.8	0.1	5000	Fisher	0.069
0.6	0.4	5000	Fisher	0.639
0.8	0.4	5000	Fisher	0.093
0.6	0.1	50000	Fisher	0.177
0.8	0.1	50000	Fisher	0.052
0.6	0.4	50000	Fisher	0.629
0.8	0.4	50000	Fisher	0.072

Comments on A:

So, according to these results, it can be seen that there is larger magnitudes for larger n and b. It means, larger sample sizes or larger scale parameters lead to a wider rejection region. For n = 5000, b = 0.8 and r = 0.4 has critical value 1.003 which is positive and larger in magnitude and for n = 50000 b = 0.6, r = 0.1 with a critical value of 1.11. This is positive and larger in magnitude.

Comments on B:

So, according to the table, when we check tests

Neyman-Person (NP) Test: Among the NP test scenarios, the performance is better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000. These scenarios have higher frequencies (Freq) compared to other combinations, indicating a higher likelihood of rejecting the null hypothesis.

For the HC_test, the performance is better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000 as almost being the same as NP test. For n=5000, NP test performed slightly better and for n=50000, they performed the same as having power = 1.0

Similar to the HC_test, the HC_mod_test, performs better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000. These scenarios have higher frequencies, indicating better performance. But both values for both n are lower than values in NP test and HC test.

The Bonferroni method, performs better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000 with having lower values than NP test. These scenarios have higher frequencies, suggesting better control of Type I error rates.

For the Chi-squared test, the performance is better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000 with having significantly lower values than NP test. These scenarios have higher frequencies, indicating better ability to detect deviations from the null hypothesis.

The Fisher test, performs better when b = 0.8 and r = 0.4 for both n = 5000 and n = 50000 with having significantly lower values than NP test as chi-squared test. These scenarios have higher frequencies, suggesting better performance in detecting differences.

So, every test performs its best when b = 0.8 and r = 0.4. Among them, the maximum power which is NP test performs the best. HC test is also performed very well which is really close to NP test.