Comparison between different local tests: Simes, Simes with Storey and Wilcoxon-Mann-Whitney

20-04-2023

The aim is to compare the performance of three closed testing procedures, which respectively use Simes local test with and without Storey estimator for the proportion of true null hypotheses and Wilcoxon-Mann-Whitney local test.

We consider a null distribution F from which inliers come and we consider outliers which come from the alternative distribution

$$F^k$$

with $k > 0$, especially if $k \in \mathbb{N}_{>0}$ we know that F^k is the distribution of the random variable defined as the maximum of k observations drawn from F. So, we consider a sample drawn from the mixture distribution

$$G = (1-\theta)F + \theta F^k$$

where $\theta \in [0,1]$ is the proportion of inliers.

Since we deal with conformal p-values, we are interested not really in the sample distribution, but rather in the scores sample distribution. In our simulation study we draw n observations for the train set, l for the calibration set and mk for the test set. All observations are drawn from a d-multivariate standard normal distribution with $d = 3$ and using the algorithm of ***isolation forest*** trained on the training samples we compute the scores for the calibration and test samples. In order to generate scores associated to outlier observations, we consider the m blocks of k observations which create the test set. For each block i with $i = 1, \ldots, m$, we draw from a Bernoulli random variable with success probability equal to θ. If the value is 1 then the i-th observation of the test set is inlier and we randomly sample its score value from the k scores of the i-th block, otherwise it is an outlier and its score value will be the maximum of the scores of the i-th block.

```r
library(mvtnorm)
library(nout)
library(isotree)
```

```
## Warning: il pacchetto 'isotree' è stato creato con R versione 4.1.3
```

```r
d_benjhoch = function(S_Y, S_X, alpha = 0.1){
  m = length(S_Y)
  n = length(S_X)
  pval = sapply(1:m, function(i) (1+sum(S_X >= S_Y[i]))/(n+1))
  d =  sum(stats::p.adjust(pval,"BH")<=alpha)
  return(d)
}









d_StoreyBH = function(S_Y, S_X, alpha = 0.1, lambda=0.5){
  m = length(S_Y)
  n = length(S_X)
  pval = sort(sapply(1:m, function(i) (1+sum(S_X >= S_Y[i]))/(n+1)), decreasing=FALSE)
  pi0Sto = sapply(1:m, function(i) (1+sum(pval>lambda))/(m*(1-lambda)))
  d =  sum(stats::p.adjust(pval,"BH")<=alpha/pi0Sto)
  return(d)
}
```

```r
scores_from_mixture = function(k, raw_scores, theta){

  if(theta>1 || theta<0){
    stop("Error: argument theta should in [0,1] interval")
  }

  ll = length(raw_scores)
  if(ll<k){
    stop("Error: length of raw_scores is smaller than k.")
  }

  quotient = ll%/%k
  remainder = ll%%k

  if(remainder != 0){
    cat("Warning: length of raw_scores is not a multiple of k. Last ",
        remainder, "elements of raw_scores will not be used.")
  }

  usable.raw_scores = raw_scores[1:(ll-remainder)]
  success = replicate(quotient, rbinom(1,1,theta))

  scores = rep(0, times = quotient)
  outlier = rep(0, times = quotient)

  for(i in 0:(quotient-1)){
    # if TRUE draw from the alternative distribution
    if(success[i+1]==T){
      scores[i+1] = max(usable.raw_scores[(i*k+1):(i*k+k)])
      outlier[i+1]=T
    }
    # if FALSE draw from the null distribution
    if(success[i+1]==F){
      scores[i+1] = sample(usable.raw_scores[(i*k+1):(i*k+k)], size=1)
    }
  }

  return(list("scores"=scores, "outlier"=outlier))
}
```

```r
simuLMPI = function(B=10^4, n, l, m, d = 3, k = 2, theta, alpha = m/(n+1)){

  train = mvtnorm::rmvnorm(n=n, mean=rep(0,d))
  iso.fo = isotree::isolation.forest(train, ndim=d, ntrees=10, nthreads=1,
                                     scoring_metric = "depth", output_score = TRUE)

  crit=critWMW(m=m, n=n, alpha=alpha)

  d_WMW = rep(0,B)
  d_Simes = rep(0,B)
  d_StoSimes = rep(0,B)
  d_BH = rep(0,B)
  d_StoBH = rep(0,B)

  for(b in 1:B){
    cal = mvtnorm::rmvnorm(n=l, mean=rep(0,d))
    te = mvtnorm::rmvnorm(n=k*m, mean=rep(0,d))

    S_cal = isotree::predict.isolation_forest(iso.fo$model, cal, type = "score")
    rawS_te = isotree::predict.isolation_forest(iso.fo$model, te, type = "score")
    gen.te.score = scores_from_mixture(k=k, raw_scores=rawS_te, theta=theta)
    S_te = gen.te.score$scores

    d_WMW[b] = d_mannwhitney(S_X=S_cal, S_Y=S_te, crit=crit)
    d_Simes[b] = d_Simes(S_X=S_cal, S_Y=S_te, alpha=alpha)
    d_StoSimes[b] = d_StoreySimes(S_X=S_cal, S_Y=S_te, alpha=alpha)
    d_BH[b] = d_benjhoch(S_X=S_cal, S_Y=S_te, alpha=alpha)
    d_StoBH[b] = d_StoreyBH(S_X=S_cal, S_Y=S_te, alpha=alpha)
  }

  discov = as.data.frame(cbind("d_BH"=d_BH>0, "d_StoBH"=d_StoBH>0, "d_Simes"=d_Simes>0,
                               "d_StoSimes"=d_StoSimes>0, "d_WMW"=d_WMW>0))
  colnames(discov) = c("BH", "BHSto", "CTSim", "CTSimSto", "CTWMW")
  res = apply(discov, MARGIN = 2, FUN = mean)

  return(list("results"=res, "discoveries"=discov, "theta"=theta, "alpha"=alpha))
}
```

K=2

```r
set.seed(321)

# Initializing parameters
B=10^4
n = 19
l = 19
m = 2
d = 3
k = 2
alpha = m/(n+1)
```

```r
thetas = c(0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1)

# Results
res = lapply(thetas, function(theta) simuLMPI(B=B, n=n, l=l, m=m, d = d,
                                              k = k, theta, alpha = m/(n+1)))

# Storing results
store_res = matrix(nrow=length(thetas), ncol = 5)
row.names = rep(NA, times=length(thetas))
for(i in 1:length(thetas)){
  row.names[i] = paste("theta =",thetas[i])
}
rownames(store_res) = row.names
colnames(store_res) = c("BH", "StoBH", "Simes", "StoSimes", "WMW")

for(i in 1:length(res)){
  store_res[i,] = res[[i]]$results
}

store_res
```

```
##                   BH  StoBH  Simes StoSimes    WMW
## theta = 0    0.0955 0.0531 0.0955   0.0044 0.0990
## theta = 0.01 0.0941 0.0520 0.0941   0.0056 0.0972
## theta = 0.03 0.0937 0.0516 0.0937   0.0053 0.0975
## theta = 0.05 0.0953 0.0509 0.0953   0.0044 0.0941
## theta = 0.07 0.1010 0.0525 0.1010   0.0054 0.0985
## theta = 0.1  0.0986 0.0536 0.0986   0.0052 0.1022
## theta = 0.3  0.1147 0.0661 0.1147   0.0063 0.1288
## theta = 0.5  0.1358 0.0851 0.1358   0.0094 0.1558
## theta = 0.7  0.1534 0.1033 0.1534   0.0112 0.1902
## theta = 0.9  0.1759 0.1273 0.1759   0.0141 0.2359
## theta = 0.95 0.1614 0.1233 0.1614   0.0144 0.2465
## theta = 0.99 0.1743 0.1263 0.1743   0.0139 0.2480
## theta = 1    0.1719 0.1290 0.1719   0.0152 0.2486
```

K=3

```r
set.seed(321)

# Initializing parameters
B=10^4
n = 19
l = 19
m = 2
d = 3
k = 3
alpha = m/(n+1)
thetas = c(0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1)

# Results
res = lapply(thetas, function(theta) simuLMPI(B=B, n=n, l=l, m=m, d = d,
                                              k = k, theta, alpha = m/(n+1)))
```

```
# Storing results
store_res = matrix(nrow=length(thetas), ncol = 5)
row.names = rep(NA, times=length(thetas))
for(i in 1:length(thetas)){
  row.names[i] = paste("theta =",thetas[i])
}
rownames(store_res) = row.names
colnames(store_res) = c("BH", "StoBH", "Simes", "StoSimes", "WMW")

for(i in 1:length(res)){
  store_res[i,] = res[[i]]$results
}

store_res
```

```
##                  BH  StoBH  Simes StoSimes    WMW
## theta = 0    0.0891 0.0461 0.0891   0.0034 0.0933
## theta = 0.01 0.0965 0.0498 0.0965   0.0042 0.0975
## theta = 0.03 0.0852 0.0491 0.0852   0.0039 0.0987
## theta = 0.05 0.1013 0.0572 0.1013   0.0056 0.1043
## theta = 0.07 0.0986 0.0550 0.0986   0.0056 0.1009
## theta = 0.1  0.1135 0.0611 0.1135   0.0063 0.1116
## theta = 0.3  0.1449 0.0901 0.1449   0.0085 0.1666
## theta = 0.5  0.1714 0.1185 0.1714   0.0143 0.2286
## theta = 0.7  0.2022 0.1591 0.2022   0.0208 0.3010
## theta = 0.9  0.2356 0.1998 0.2356   0.0275 0.3673
## theta = 0.95 0.2457 0.2073 0.2457   0.0278 0.3866
## theta = 0.99 0.2506 0.2185 0.2506   0.0327 0.4063
## theta = 1    0.2485 0.2145 0.2485   0.0283 0.4038
```

K=5

```
set.seed(321)

# Initializing parameters
B=10^4
n = 19
l = 19
m = 2
d = 3
k = 5
alpha = m/(n+1)
thetas = c(0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1)

# Results
res = lapply(thetas, function(theta) simuLMPI(B=B, n=n, l=l, m=m, d = d,
                                              k = k, theta, alpha = m/(n+1)))

# Storing results
store_res = matrix(nrow=length(thetas), ncol = 5)
row.names = rep(NA, times=length(thetas))
for(i in 1:length(thetas)){
  row.names[i] = paste("theta =",thetas[i])
}
rownames(store_res) = row.names
colnames(store_res) = c("BH", "StoBH", "Simes", "StoSimes", "WMW")

for(i in 1:length(res)){
  store_res[i,] = res[[i]]$results
}

store_res
```

```
##                   BH  StoBH  Simes StoSimes    WMW
## theta = 0    0.0917 0.0470 0.0917   0.0028 0.0909
## theta = 0.01 0.0999 0.0541 0.0999   0.0054 0.0975
## theta = 0.03 0.0936 0.0522 0.0936   0.0054 0.0985
## theta = 0.05 0.1049 0.0569 0.1049   0.0053 0.1077
## theta = 0.07 0.1092 0.0610 0.1092   0.0066 0.1165
## theta = 0.1  0.1241 0.0704 0.1241   0.0076 0.1294
## theta = 0.3  0.1873 0.1209 0.1873   0.0154 0.2074
## theta = 0.5  0.2404 0.1755 0.2404   0.0247 0.3053
## theta = 0.7  0.2842 0.2351 0.2842   0.0421 0.4240
## theta = 0.9  0.3315 0.3027 0.3315   0.0541 0.5548
## theta = 0.95 0.3436 0.3205 0.3436   0.0579 0.5926
## theta = 0.99 0.3750 0.3554 0.3750   0.0688 0.6286
## theta = 1    0.3762 0.3619 0.3762   0.0674 0.6373
```

K=10

```
set.seed(321)

# Initializing parameters
B=10^4
n = 19
l = 19
m = 2
d = 3
k = 10
alpha = m/(n+1)
thetas = c(0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1)

# Results
res = lapply(thetas, function(theta) simuLMPI(B=B, n=n, l=l, m=m, d = d,
                                              k = k, theta, alpha = m/(n+1)))

# Storing results
store_res = matrix(nrow=length(thetas), ncol = 5)
row.names = rep(NA, times=length(thetas))
for(i in 1:length(thetas)){
  row.names[i] = paste("theta =",thetas[i])
}
rownames(store_res) = row.names
colnames(store_res) = c("BH", "StoBH", "Simes", "StoSimes", "WMW")

for(i in 1:length(res)){
  store_res[i,] = res[[i]]$results
}

store_res
```

```
##                   BH  StoBH  Simes StoSimes    WMW
## theta = 0    0.0877 0.0472 0.0877   0.0035 0.0899
## theta = 0.01 0.0991 0.0524 0.0991   0.0050 0.0952
## theta = 0.03 0.1033 0.0559 0.1033   0.0048 0.1040
## theta = 0.05 0.1221 0.0658 0.1221   0.0071 0.1200
## theta = 0.07 0.1349 0.0752 0.1349   0.0099 0.1260
## theta = 0.1  0.1426 0.0825 0.1426   0.0094 0.1374
## theta = 0.3  0.2490 0.1603 0.2490   0.0278 0.2563
## theta = 0.5  0.3495 0.2601 0.3495   0.0584 0.3974
## theta = 0.7  0.4467 0.3730 0.4467   0.0931 0.5677
## theta = 0.9  0.5331 0.5008 0.5331   0.1415 0.7756
## theta = 0.95 0.5608 0.5426 0.5608   0.1557 0.8226
## theta = 0.99 0.5928 0.5858 0.5928   0.1701 0.8784
## theta = 1    0.5739 0.5723 0.5739   0.1616 0.8823
```

K=15

```
set.seed(321)

# Initializing parameters
B=10^3
n = 19
l = 19
m = 2
d = 3
k = 15
alpha = m/(n+1)
thetas = c(0, 0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1)

# Results
res = lapply(thetas, function(theta) simuLMPI(B=B, n=n, l=l, m=m, d = d,
                                              k = k, theta, alpha = m/(n+1)))

# Storing results
store_res = matrix(nrow=length(thetas), ncol = 5)
row.names = rep(NA, times=length(thetas))
for(i in 1:length(thetas)){
  row.names[i] = paste("theta =",thetas[i])
}
rownames(store_res) = row.names
colnames(store_res) = c("BH", "StoBH", "Simes", "StoSimes", "WMW")

for(i in 1:length(res)){
  store_res[i,] = res[[i]]$results
}

store_res
```

```
##                  BH StoBH Simes StoSimes    WMW
## theta = 0    0.084 0.045 0.084    0.005 0.085
## theta = 0.01 0.109 0.057 0.109    0.005 0.096
## theta = 0.03 0.117 0.059 0.117    0.003 0.095
## theta = 0.05 0.130 0.068 0.130    0.009 0.115
## theta = 0.07 0.140 0.075 0.140    0.007 0.131
## theta = 0.1  0.146 0.083 0.146    0.014 0.141
## theta = 0.3  0.287 0.201 0.287    0.040 0.284
## theta = 0.5  0.422 0.306 0.422    0.082 0.420
## theta = 0.7  0.507 0.420 0.507    0.133 0.592
## theta = 0.9  0.643 0.611 0.643    0.205 0.841
## theta = 0.95 0.665 0.646 0.665    0.202 0.875
## theta = 0.99 0.701 0.695 0.701    0.260 0.944
## theta = 1    0.706 0.705 0.706    0.254 0.960
```