Comparison of testCompareR, DTComPair and compbdt

Kyle J. Wilson

## Introduction

This script was used during testCompareR’s development to benchmark testCompareR against the only available R package, DTComPair, which performs a similar function, and its open-source predecessor, compbdt. The full manuscript has further details on the development, mathematical bases and use cases for testCompareR.

## Preparing the data

While testCompareR can accept a data frame, matrix or tibble as its argument, both DTComPair and compbdt require the user to pre-process the data into the accepted format for their functions. To reproduce this script, it is necessary to load the compbdt function into memory. Here, that is achieved through source().

Loading required package: PropCIs

here() starts at /Users/kyle/Dropbox/LiverpoolUniversity\_MLW/testCompareR

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':  
  
 filter, lag

The following objects are masked from 'package:base':  
  
 intersect, setdiff, setequal, union

Loading required package: viridisLite

Attaching package: 'scales'

The following object is masked from 'package:viridis':  
  
 viridis\_pal

$testCompareR  
[1] "testCompareR" "stats" "graphics" "grDevices" "utils"   
[6] "datasets" "methods" "base"   
  
$DTComPair  
 [1] "DTComPair" "PropCIs" "testCompareR" "stats" "graphics"   
 [6] "grDevices" "utils" "datasets" "methods" "base"   
  
$here  
 [1] "here" "DTComPair" "PropCIs" "testCompareR" "stats"   
 [6] "graphics" "grDevices" "utils" "datasets" "methods"   
[11] "base"   
  
$microbenchmark  
 [1] "microbenchmark" "here" "DTComPair" "PropCIs"   
 [5] "testCompareR" "stats" "graphics" "grDevices"   
 [9] "utils" "datasets" "methods" "base"   
  
$dplyr  
 [1] "dplyr" "microbenchmark" "here" "DTComPair"   
 [5] "PropCIs" "testCompareR" "stats" "graphics"   
 [9] "grDevices" "utils" "datasets" "methods"   
[13] "base"   
  
$ggplot2  
 [1] "ggplot2" "dplyr" "microbenchmark" "here"   
 [5] "DTComPair" "PropCIs" "testCompareR" "stats"   
 [9] "graphics" "grDevices" "utils" "datasets"   
[13] "methods" "base"   
  
$viridis  
 [1] "viridis" "viridisLite" "ggplot2" "dplyr"   
 [5] "microbenchmark" "here" "DTComPair" "PropCIs"   
 [9] "testCompareR" "stats" "graphics" "grDevices"   
[13] "utils" "datasets" "methods" "base"   
  
$tidyr  
 [1] "tidyr" "viridis" "viridisLite" "ggplot2"   
 [5] "dplyr" "microbenchmark" "here" "DTComPair"   
 [9] "PropCIs" "testCompareR" "stats" "graphics"   
[13] "grDevices" "utils" "datasets" "methods"   
[17] "base"   
  
$scales  
 [1] "scales" "tidyr" "viridis" "viridisLite"   
 [5] "ggplot2" "dplyr" "microbenchmark" "here"   
 [9] "DTComPair" "PropCIs" "testCompareR" "stats"   
[13] "graphics" "grDevices" "utils" "datasets"   
[17] "methods" "base"

Now that we have loaded each of the packages / functions, we can run some sample data through each of them. The cass data is provided within the testCompareR package and here is restructured according to the requirements of each program.

# compute results for testCompareR  
results <- compareR(cass, dp=10)  
  
# create a function which mirrors the functionality of compareR() for DTComPair  
DTComPair <- function(dat) {  
  
 dtc <- tab.paired(d = dat[,3], y1 = dat[,1], dat[,2])  
  
 return(list(  
 sum = acc.paired(dtc), # returns point estimates and CIs  
 acc = sesp.mcnemar(dtc), # compares diagnostic accuracies with McNemars  
 pv = pv.wgs(dtc), # compares PVs with weighted generalised score statistic  
 lr = dlr.regtest(dtc)) # compares LRs with regression model  
 )  
   
}  
  
# compute results for DTComPair  
results.dt <- DTComPair(cass)  
  
# pre-process data for compbdt using internal function from testCompareR  
dat.compbdt <- testCompareR:::values.2test(cass)  
  
# manually run compbdt and extract numbers (changed decip to 6 in function)  
compbdt(dat.compbdt$s11, dat.compbdt$s10,  
 dat.compbdt$s01, dat.compbdt$s00,  
 dat.compbdt$r11, dat.compbdt$r10,  
 dat.compbdt$r01, dat.compbdt$r00)

PREVALENCE OF THE DISEASE   
  
Estimated prevalence of the disease is 69.80482 % and its standard error is 1.555616   
  
95 % confidence interval for the prevalence of the disease is ( 66.68067 % ; 72.76765 %)   
  
 COMPARISON OF THE ACCURACIES (SENSITIVITIES AND SPECIFICITIES)   
  
Estimated sensitivity of Test 1 is 82.56579 % and its standard error is 1.538684   
  
95 % confidence interval for the sensitivity of Test 1 is ( 79.36292 % ; 85.38937 %)   
  
Estimated sensitivity of Test 2 is 91.11842 % and its standard error is 1.153709   
  
95 % confidence interval for the sensitivity of Test 1 is ( 88.61012 % ; 93.14782 %)   
  
Estimated specificity of Test 1 is 74.14449 % and its standard error is 2.699841   
  
95 % confidence interval for the specificity of Test 1 is ( 68.55724 % ; 79.08695 %)   
  
Estimated specificity of Test 2 is 74.90494 % and its standard error is 2.673446   
  
95 % confidence interval for the specificity of Test 1 is ( 69.35805 % ; 79.78674 %)   
  
  
Wald test statistic for the global hypothesis test H0: (Se1 = Se2 and Sp1 = Sp2) is 25.662   
  
 Global p-value is 3e-06   
  
 Applying the global Wald test (to an alpha error of 5 %), we reject the hypothesis H0: (Se1 = Se2 and Sp1 = Sp2)   
  
 Estimated power (to an alpha error of 5 %) is 99.8 %   
  
 Investigation of the causes of significance:   
  
 McNemar test statistic (with cc) for H0: Se1 = Se2 is 23.64545 and the two-sided p-value is 0   
  
 McNemar test statistic (with cc) for H0: Sp1 = Sp2 is 0.011111 and the two-sided p-value is 0.991135   
  
 Applying the Holm method (to an alpha error of 5 %), we reject the hypothesis H0: Se1 = Se2 and we do not reject the hypothesis H0: Sp1 = Sp2   
  
 Sensitivity of Test 2 is significantly greater than sensitivity of Test 1   
  
 95 % confidence interval for the difference Se2 - Se1 is ( 5.192182 % ; 11.857 %)   
  
 COMPARISON OF THE LIKELIHOOD RATIOS   
  
Estimated positive LR of Test 1 is 3.193353 and its standard error is 0.33872   
  
95 % confidence interval for the positive LR of Test 1 is ( 2.610218 ; 3.952402 )   
  
Estimated positive LR of Test 2 is 3.630931 and its standard error is 0.389536   
  
95 % confidence interval for the positive LR of Test 1 is ( 2.96151 ; 4.505412 )   
  
Estimated negative LR of Test 1 is 0.235138 and its standard error is 0.022449   
  
95 % confidence interval for the negative LR of Test 1 is ( 0.194711 ; 0.283254 )   
  
Estimated negative LR of Test 2 is 0.118571 and its standard error is 0.015973   
  
95 % confidence interval for the negative LR of Test 2 is ( 0.090098 ; 0.153236 )   
  
  
Test statistic for the global hypothesis test H0: (PLR1 = PLR2 and NLR1 = NLR2) is 23.43805   
  
 Global p-value is 8e-06   
  
 Applying the global hypothesis test (to an alpha error of 5 %), we reject the hypothesis H0: (PLR1 = PLR2 and NLR1 = NLR2)   
  
 Estimated power (to an alpha error of 5 %) is 99.78 %   
  
 Investigation of the causes of significance:   
  
 Test statistic for H0: PLR1 = PLR2 is 0.898025 and the two-sided p-value is 0.369172   
  
 Test statistic for H0: NLR1 = NLR2 is 4.662817 and the two-sided p-value is 3e-06   
  
 Applying the Holm method (to an alpha error of 5 %), we do not reject the hypothesis H0: PLR1 = PLR2 and we reject the hypothesis H0: NLR1 = NLR2   
  
 Negative likelihood ratio of Test 1 is significantly greater than negative likelihood ratio of Test 2   
  
 95 % confidence interval for the ratio NLR1 / NLR2 is ( 1.412383 ; 2.553809 )   
  
 COMPARISON OF THE PREDICTIVE VALUES   
  
Estimated positive PV of Test 1 is 88.07018 % and its standard error is 1.357669   
  
95 % confidence interval for the positive PV of Test 1 is ( 85.16952 % ; 90.49807 %)   
  
Estimated positive PV of Test 2 is 89.35484 % and its standard error is 1.238624   
  
95 % confidence interval for the positive PV of Test 2 is ( 86.69779 % ; 91.56234 %)   
  
Estimated negative PV of Test 1 is 64.78405 % and its standard error is 2.753091   
  
95 % confidence interval for the negative PV of Test 1 is ( 59.24617 % ; 69.97634 %)   
  
Estimated negative PV of Test 2 is 78.48606 % and its standard error is 2.593698   
  
95 % confidence interval for the negative PV of Test 2 is ( 73.02445 % ; 83.15112 %)   
  
  
Wald test statistic for the global hypothesis test H0: (PPV1 = PPV2 and NPV1 = NPV2) is 25.94449   
  
 Global p-value is 2e-06   
  
 Applying the global hypothesis test (to an alpha error of 5 %), we reject the hypothesis H0: (PPV1 = PPV2 and NPV1 = NPV2)   
  
 Estimated power (to an alpha error of 5 %) is 99.26 %   
  
 Investigation of the causes of significance:   
  
 Weighted generalized score statistic for H0: PPV1 = PPV2 is 0.807058 and the two-sided p-value is 0.368992   
  
 Weighted generalized score statistic for H0: NPV1 = NPV2 is 22.50225 and the two-sided p-value is 2e-06   
  
 Applying the Holm method (to an alpha error of 5 %), we do not reject the hypothesis H0: PPV1 = PPV2 and we reject the hypothesis H0: NPV1 = NPV2   
  
 Negative PV of Test 2 is significantly greater than negative PV of Test 1   
  
 95 % confidence interval for the difference NPV2 - NPV1 is ( 8.040664 % ; 19.36334 %)

Next, we can create a large list containing all of the values which make up Table 4 in the testCompareR manuscript. This code is long and cumbersome, so is not produced in the Word document, but is available within the QMD file. At the end the list is output to benchmarking/comparison.rda.

Next, we can run the efficiency tests. I have run testCompareR from within a function wrapper, in case that offers marginal gains over DTComPair.

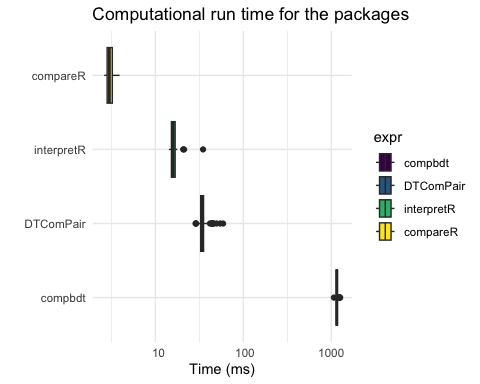
# pre-processing of data for separate package  
dat.compareR <- cass  
dat.DTComPair <- data.frame(cass$angio, cass$exercise, cass$cp)  
dat.compbdt <- testCompareR:::values.2test(cass)  
  
# run compareR from within a function to make similar to DTComPair  
test <- function(dat) {  
 compareR(dat)  
}  
  
# run all functions from DTComPair to create similar readout to compareR  
DTComPair <- function(dat) {  
 dtc <- tab.paired(d = dat[,3], y1 = dat[,1], dat[,2], data = dat)  
 acc.paired(dtc)  
 sesp.mcnemar(dtc)  
 pv.wgs(dtc)  
 dlr.regtest(dtc)  
}  
  
# calculate times for 1) compareR 2) compareR+interpretR 3) DTComPair 4) compbdt  
efficiency <- microbenchmark(  
 compareR = test(dat.compareR),  
 interpretR = interpretR(test(dat.compareR)),  
 DTComPair = DTComPair(dat.DTComPair),  
 compbdt = compbdt(dat.compbdt$s11, dat.compbdt$s10,  
 dat.compbdt$s01, dat.compbdt$s00,  
 dat.compbdt$r11, dat.compbdt$r10,  
 dat.compbdt$r01, dat.compbdt$r00),  
 times = 100  
)  
  
save(efficiency, file = here("benchmarking/comparative\_efficiencies.rda"))

Now we want to investigate the reason DTComPair is slower than testCompareR.

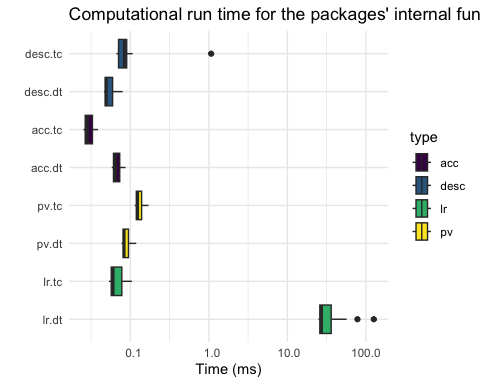
# investigate efficiency of DTComPair  
d <- as.vector(cass$angio)  
y1 <- as.vector(cass$exercise)  
y2 <- as.vector(cass$cp)  
dtc <- tab.paired(d = d, y1 = y1, y2 = y2)  
  
vals <- testCompareR:::values.2test(cass)  
  
descriptive <- function(vals) {  
 testCompareR:::conf.prev(vals)  
 testCompareR:::conf.acc(vals)  
 testCompareR:::conf.pv(vals)  
 testCompareR:::conf.lr(vals)  
}  
  
# individual functions vs. internal functions  
internal.efficiency <- microbenchmark(  
 desc.tc = descriptive(vals),  
 desc.dt = acc.paired(dtc),  
 acc.tc = testCompareR:::output.acc(vals),  
 acc.dt = sesp.mcnemar(dtc),  
 pv.tc = testCompareR:::output.pv(vals),  
 pv.dt = pv.wgs(dtc),  
 lr.tc = testCompareR:::output.lr(vals),  
 lr.dt = dlr.regtest(dtc),  
 times = 100  
)  
  
internal.efficiency <- internal.efficiency %>% arrange(expr)  
internal.efficiency$type <- c(rep("desc", 200), rep("acc", 200), rep("pv", 200), rep("lr", 200))  
  
save(internal.efficiency, file = here("benchmarking/internal\_efficiencies.rda"))

Here I have run the tests on a different machine to when I prepared the manuscript, but as the results are very similar and the specifications have been clearly described in the paper there is no need to update the figures or tables. Nevertheless, for demonstration purposes I show how the plots were generated.

load(here("benchmarking/comparative\_efficiencies.rda"))  
load(here("benchmarking/internal\_efficiencies.rda"))  
  
efficiency$expr <- factor(efficiency$expr,   
 levels = c("compbdt", "DTComPair", "interpretR",  
 "compareR"))  
efficiency$time <- efficiency$time / 1000000  
  
ggplot(efficiency, aes(x = time, y = expr, fill = expr)) +  
 geom\_boxplot() +   
 theme\_minimal() +  
 scale\_fill\_viridis(discrete = TRUE) +  
 scale\_x\_continuous(trans = "log10",   
 labels = function(x) format(x, scientific = FALSE)) +  
 labs(x = "Time (ms)", y = "",   
 title = "Computational run time for the packages")



internal.efficiency$expr <- factor(internal.efficiency$expr,   
 levels = c("lr.dt", "lr.tc", "pv.dt", "pv.tc",   
 "acc.dt", "acc.tc", "desc.dt", "desc.tc")  
 )  
internal.efficiency$time <- internal.efficiency$time / 1000000  
  
ggplot(internal.efficiency, aes(x = time, y = expr, fill = type)) +  
 geom\_boxplot() +   
 theme\_minimal() +  
 scale\_fill\_viridis(discrete = TRUE) +  
 scale\_x\_continuous(trans = "log10",   
 labels = function(x) format(x, scientific = FALSE)) +  
 labs(x = "Time (ms)", y = "",   
 title = "Computational run time for the packages' internal functions")



## How efficiency changes with data size

To test the efficiency with different sized data sets I duplicated the a simulated data set to create progressively larger data sets. I scaled linearly, in order to see if the computational time also scales linearly.

It is not useful to run compbdt this many times, as it is already demonstrably slower and the pre-processing has to be handled by testCompareR anyway.

First, we can make a simulated data set using the internal dataframeR function. I want to create a small increase in sensitivity which costs a lot in terms of specificity.

dat <- testCompareR:::dataframeR(800,12,8,180,300,10,90,600)

Next, we set up the efficiency testing.

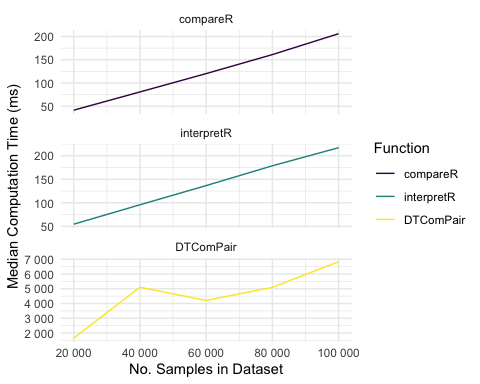
dat.compareR <- dat  
dat.compareR.list <- list()  
  
for(n in c(10,20,30,40,50)) {   
 df <- do.call("rbind", replicate(n, dat.compareR, simplify = FALSE))   
 dat.compareR.list <- append(dat.compareR.list, list(df))   
}  
  
dat.DTComPair <- data.frame(dat$gold, dat$test1, dat$test2)  
dat.DTComPair.list <- list()  
  
for(n in c(10,20,30,40,50)) {   
 df <- do.call("rbind", replicate(n, dat.compareR, simplify = FALSE))   
 dat.DTComPair.list <- append(dat.DTComPair.list, list(df))   
}

Finally, we perform the efficiency testing. This is computationally intensive and will take some minutes to run.

efficiencies <- list()  
  
for(i in 1:length(dat.compareR.list)) {  
   
 test <- function(dat) {  
 compareR(dat)  
 }  
   
 efficiency <- microbenchmark(  
 compareR = test(dat.compareR.list[[i]]),  
 interpretR = interpretR(test(dat.compareR.list[[i]])),  
 DTComPair = DTComPair(dat.DTComPair.list[[i]]),  
 times = 25)  
   
 efficiencies <- append(efficiencies, efficiency)  
   
}  
  
names(efficiencies) <- paste(names(efficiencies), c(1,1,2,2,3,3,4,4,5,5),  
 sep = "\_")  
  
save(efficiencies, file = here("benchmarking/scaled\_efficiencies.rda"))

Finally, we can plot the outcome.

load(here("benchmarking/scaled\_efficiencies.rda"))  
  
df <- as.data.frame(efficiencies)  
  
df\_long <- df %>%  
 pivot\_longer(  
 cols = everything(),  
 names\_to = c(".value", "iteration"),  
 names\_pattern = "(expr|time)\_(\\d+)"  
 )  
  
df\_long$iteration <- as.numeric(df\_long$iteration) \* nrow(dat) \* 10  
  
df\_long <- df\_long %>%   
 group\_by(iteration, expr) %>%  
 mutate(  
 median\_c = median(time) / 1e+6  
 )  
  
ggplot(df\_long, aes(x = iteration, y = median\_c, group = expr, colour = expr)) +  
 geom\_line() +  
 facet\_wrap(~expr, scales = "free\_y", nrow = 3) +  
 scale\_color\_viridis(discrete = TRUE) +  
 scale\_x\_continuous(labels = label\_number(scale = 1)) +  
 scale\_y\_continuous(labels = label\_number(scale = 1)) +  
 labs(x = "No. Samples in Dataset", y = "Median Computation Time (ms)", color = "Function") +  
 theme\_minimal()



## Conclusions

Here we have shown that testCompareR achieves comparable results to DTComPair and compbdt, that it performs more quickly, even when wrapped in the interpretR function and that computation time scales linearly with data size, outperforming DTComPair on large datasets.