Personal work – Differential analysis and multiple testing

Probabilities and statistics for modelling 1 (STAT1)

Jacques van Helden

2019-10-26

Contents

Introduction

The personal work will consist in developing a workflow to run an experimental benchmark of differential analysis applied to thousands of features.

The goal will be to acquire an empirical feeling of the impact of different parameters on the performances of differential analysis tests.

To this purpose, we will work with artificial data produced by generating random numbers

- following different distribiutions of probabilities (normal, Poisson),
- with different differences between the population means
- with equal or distinct variances

We will then run different tests and compare their performances by measuring the power and false positive rates.

Performances of the tests

We will measure the performances of a test by running $r = 10,000$ times under H_0, and $r = 10,000$ times under H_1.

- count the number of FP, TP, FN, TN
- summarise the result in a confusion table
- compute the derived statistics: FPR, FDR and Sn

Questions

1. Impact of the difference between the means

We test the impact of the actual difference between population means on the performances of the Student test. For this, generate random datasets compliant with the assumptions (normality, homoscedasticity) but with a set of increasing differences between sample means.

In all cases, we will set the standard deviation to $\sigma = 1$.

μ_1	μ_2	δ	Comment
0	0	0	H_0
-0.05	0.05	0.1	H_1
-0.1	0.1	0.2	H_1
-0.25	0.25	0.5	H_1
-0.5	0.5	1	H_1
-1	1	2	H_1

Maybe other values if you have time and if it brings additional insight.

2. Impact of sample size

Generate random numbers following a normal distribution, with equal variances, but with a series of different sample sizes ($n = 2, 4, 16, 64$).

Run Student test and evaluate the impact of sample size on the rate of false positive and power of the test.

3. Impact of variance

Generate random numbers following a normal distribution, with equal sample sizes ($n_1 = n_2 = 10$), but with a series of different standard deviations ($\sigma = 0.1, 1, 10, 100$).

Run Student test and evaluate the impact of sample size on the rate of false positive and power of the test.

4. Impact of non-normality on Student test

Generate random numbers following either a normal or a Poisson distribution.

1a. Small samples: $n_1 = n_2 = 4$ 1b. Large samples: $n_1 = n_2 = 50$

5. Impact of heteroscedasticity on Student test

Generate random numbers following a normal distribution with either equal or unequal variances (test different values). Run Student test on both data types, and measure the sensitivity and power of the respective tests.

6. Impact of homoscedasticity on Welch test

Generate random numbers following a normal distribution with either equal or unequal variances (test different values). Run Welch test on both data types, and measure the sensitivity and power of the respective

tests.

7. Parametric versus non-parametric tests

Generate random numbers following a normal and some non-normal distribution of your choice (e.g. uniform). Run a parametric and a non-parametric test and compare the performances in terms fo power and false positive rate.

Assignment of the tasks

Each student will develop an analysis to test one particular question.

Student	Question
JvH + you	1. Impact of the difference between the means

Expected results

- Descriptive statistics (explor your data):
  - graphical representations of the data (use whatever you like: histograms, boxplots, violin plots, dot plots, ...)
  - descriptive parameters of your samples (distribution of sample means, sample sd, ...)
- Histogram of the p-values (20 bins) for all the cases under H_0, and under H_1, resp.
- Volcano plot for all the cases under H_0, and under H_1, resp.
- Indicators: FPR, FDR, Sn, $E-value$ for threshold values $alpha = 0.05$, $alpha = 0.01$, , $alpha = 0.001$

Report format

- R markdown (must be able to run on my machine)
- either a pdf or a self-contained HTML (preferred because easier to browse tables are better formatted)

Structure of the report

- Introduction: explain the question addressed in your specific report, and the expected outcome of what you are testing (do you expect an increase of FP ? a loss in sensitivity ? ...).
- Setting of the experiment: write explicitly the configuration of your tests (distributions, parameters)
- Results and interpretation: summarize your results with figures and tables, document the figures and tables with detailed legends, and add an interpretation of what we see in the results section.
- Summary and conclusion: summarize the results in 2-3 sentences and conclude about the effect you wanted to test.

Recommendations

Coding recommendations

1. Choose a consistent coding style, consistent with a reference style guide (e.g. Google R Syle Guide)
2. Name each chunk of R code
3. Define your variables with explicit names (sigma, mu rather than a, b, c, ...)
4. Comment your code

- indicate what each variable represents
- before each segment of code, explain what it will do

5. Ensure consistency between the code and the report → inject the actual values of the R variables in the markdown.

Scientific recommendations

1. Explicitly formulate the statistical hypotheses before running a test.
2. Discuss the assumptions underlying the test: are they all fulfilled? If not explain why (e.g. because we want to test the impact of this parameter, ...)

Tutorial

Define your parameters

We draw samples from random data generated following normal distributions with the following parameters. For this we use R `rnorm()` function.

```
## Define parameters
mu1 <- 0 # mean of the first population
mu2 <- 0 # mean of the second population
sigma1 <- 1 # standard deviation of the first population
sigma2 <- 1 # standard deviation of the second population
n1 <- 10 # sample size for the first group
n2 <- 10 # ssample size for second group

alpha <- 0.05 # Probability of the first kind error, used as threshold on p-value

r <- 10000 # Number of replicates of the test

## A trick for this report: let us set an arbitrary seed in order to ensure consistency between the int
##
## I take 42, Deep Thought's Answer to the Ultimate Question of Life, the Universe, and Everything (42)
set.seed(seed = 42)
```

Parameter	Value	Description
μ_1	0	Mean of the first population
μ_2	0	Mean of the second population
σ_1	1	Standard deviation of the first population
σ_2	1	Standard deviation of the second population
n_1	10	Sample size for the first group
n_2	10	Sample size for the second group

Generate one test set (1 sample from each population)

The table below shows the values of the two samples (rounded to two decimals for the sake of readability).

```
## Generate two vectors containing the values for sample 1 and sample 2, resp.
x1 <- rnorm(n = n1, mean = mu1, sd = sigma1) ## sample 1 values
x2 <- rnorm(n = n2, mean = mu2, sd = sigma2) ## sample 2 values

kable(t(data.frame(x1 = round(digits = 2, x1),
                   x2 = round(digits = 2, x2))),
      col.names = 1:n1
```

```
)
```

	1	2	3	4	5	6	7	8	9	10
x1	1.37	-0.56	0.36	0.63	0.40	-0.11	1.51	-0.09	2.02	-0.06
x2	1.30	2.29	-1.39	-0.28	-0.13	0.64	-0.28	-2.66	-2.44	1.32

Run Student test

Since we are interested by differences in either directions, we run a two-tailed test.

Hypotheses:

$$H_0 : \mu_1 = \mu 2$$

$$H_1 : \mu_1 \neq \mu 2$$

```
## Run Sudent t test on one pair of samples
t.result <- t.test(
  x = x1, y = x2,
  alternative = "two.sided", var.equal = TRUE)

## Print the result of the t test
print(t.result)
```

```
    Two Sample t-test

data:  x1 and x2
t = 1.2268, df = 18, p-value = 0.2357
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.506473  1.927980
sample estimates:
 mean of x  mean of y
 0.5472968 -0.1634567
```

```
## Compute some additional statistics about the samples
mean1 <- mean(x1) ## Mean of sample 1
mean2 <- mean(x2) ## Mean of sample 2
d <- mean2 - mean1 ## Difference between sample means
```

Interpret the result

The difference between sample means was $d = -0.71$.

The t test computed the t statistics, which standardizes this observed distance between sample means relative to the estimated variance of the population, and to the sample sizes. With the random numbers generated above, the value is $t_{obs} = 1.2268$.

The corresponding p-value is computed as the sum of the area of the left and right tails of the Student distribution, with $\nu = n_1 + n_2 - 2 = 18$ degrees of freedom. It indicates the probability of obtaining by chance – under th null hypothesis – a result at least as extreme as the one we observed.

In our case, we obtain $p = P(T > |t_{obs}| = P(T > 1.2268) = 0.236$. This is higher than our threshold $alpha = 0.05$. We thus accept the null hypothesis.

Replicating the test 10,000 times with a loop

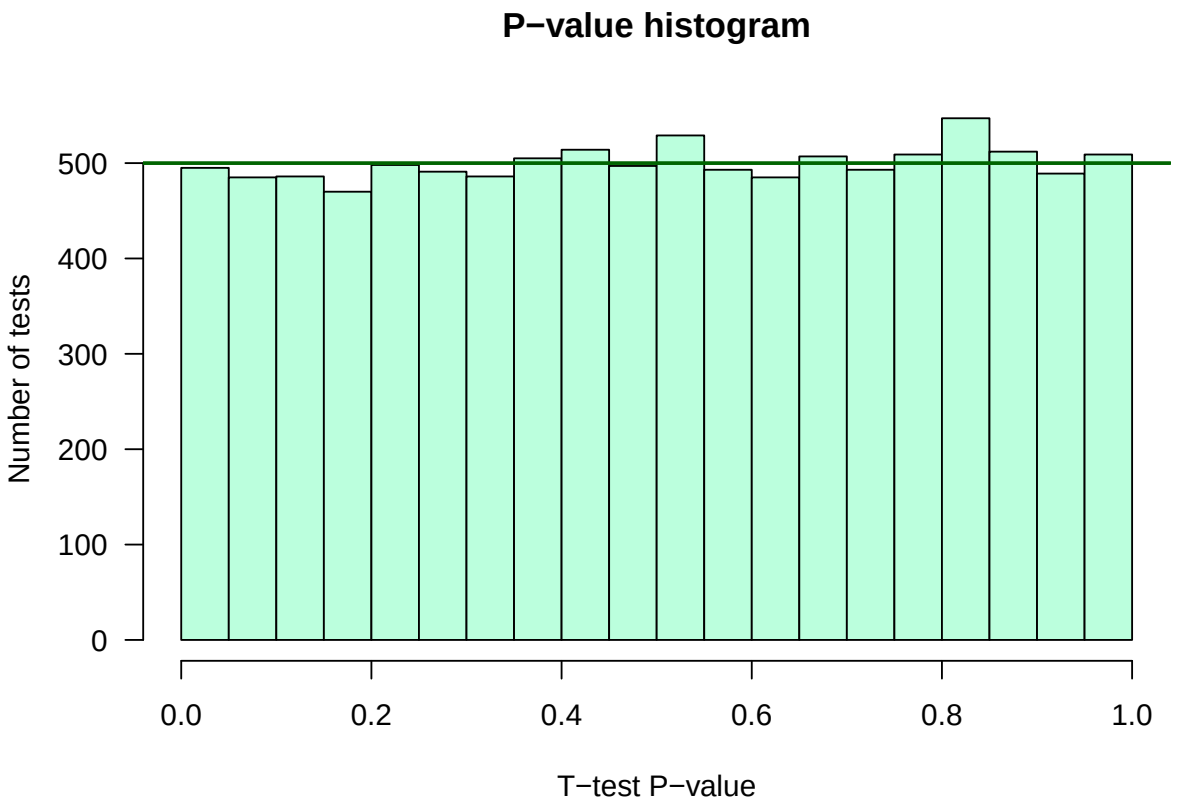
In R, loops are quite inefficient, and it is generally recommended to directly run the computations on whole vectors (R has been designed to be efficient for this), or to use specific functions in order to apply a given function each row / column of a table, or to each element of a list.

For the sake of simplicity, we will first show how to implement a simple but inefficient code with a loop. In the advanced course (STATS2) will see how to optimize the speed with the `apply()` function.

```
## Define the statistics we want to collect
result.columns <- c("i", "m1", "m2", "diff", "statistic", "p.value")

## Instantiate a result table to store the results
result.table <- data.frame(matrix(nrow = r, ncol = length(result.columns)))
colnames(result.table) <- result.columns # set the column names
# View(result.table) ## Check the table: it contians NA values

## Iterate random number sampling followed by t-tests
for (i in 1:r) {
  ## Generate two vectors containing the values for sample 1 and sample 2, resp.
  x1 <- rnorm(n = n1, mean = mu1, sd = sigma1) ## sample 1 values
  x2 <- rnorm(n = n2, mean = mu2, sd = sigma2) ## sample 2 values

  ## Run the t test
  t.result <- t.test(
    x = x1, y = x2,
    alternative = "two.sided", var.equal = TRUE)
  # names(t.result)

  ## Collect the selected statistics in the result table
  result.table[i, "i"] <- i
  result.table[i, "statistic"] <- t.result["statistic"]
  result.table[i, "p.value"] <- t.result["p.value"]

  ## Compute some additional statistics about the samples
  result.table[i, "m1"] <- mean(x1) ## Mean of sample 1
  result.table[i, "m2"] <- mean(x2) ## Mean of sample 2
  result.table[i, "diff"] <- mean2 - mean1 ## Difference between sample means

}

## View(result.table)
```

P-value histogram

```
## Draw an histogram of p-values with 20 bins
hist(result.table$p.value, breaks = 20,
     col = "#BBFFDD", las = 1,
     main = "P-value histogram",
     xlab = "T-test P-value",
     ylab = "Number of tests")

## Draw a horizontal line indicating the number of tests per bin that would be expected under null hypo
abline(h = r / 20, col = "darkgreen", lwd = 2)
```



Creating a function to reuse the same code with different parameters

Depending on the selected task in the assignments above, we will run different tests with different parameters and compare the results. The most rudimentary way to do this is top copy-paste the chunk of code above for each test and set of parameters required for the assigned tasks.

However, having several copies of an almost identical block of code is a very bad pracrice in programming, for several reasons

- lack of readability: the code rapidly becomes very heavy;
- difficulty to maintain: any modification has to be done on each copy of the chunk of code;
- risk for consistency: this is a source of inconsistency, because at some moment we will modify one copy and forget another one.

A better practice is to define a **function** that encapsulates the code, and enables to modify the parameters by passing them as **arguments*. Hereafter we define a function that

- takes the parameters of the analysis as arguments
  - population means μ_1 and μ_2,
  - population standard deviations σ_1 and σ_2,
  - sample sizes n_1 and n_2,
  - number of iterations r.
- runs r iterations of the t-test with 2 random samples,
- returns the results in a table with one row per iteration, and one column per resulting statistics (observed t score, p-value, difference between means, ...);

```r
## Define a function that runs r iterations of the t-test.
iterate.t.test <- function(mu1 = -0.5, ## Mean of the first population
                           mu2 = 0.5,  ## Mean of the second population
                           sigma1 = 1, ## Standard deviation of the first population
                           sigma2 = 1,  ## Standard deviation of the second population
                           n1 = 10,    ## First sample size
                           n2 = 10,    ## Second sample size
                           r = 10000  ## Iterations
                           ) {

  ## Define the statistics we want to collect
  result.columns <- c("i", "m1", "m2", "diff", "statistic", "p.value")

  ## Instantiate a result table to store the results
  result.table <- data.frame(matrix(nrow = r, ncol = length(result.columns)))
  colnames(result.table) <- result.columns # set the column names
  # View(result.table) ## Check the table: it contians NA values

  ## Iterate random number sampling followed by t-tests
  for (i in 1:r) {
    ## Generate two vectors containing the values for sample 1 and sample 2, resp.
    x1 <- rnorm(n = n1, mean = mu1, sd = sigma1) ## sample 1 values
    x2 <- rnorm(n = n2, mean = mu2, sd = sigma2) ## sample 2 values

    ## Run the t test
    t.result <- t.test(
      x = x1, y = x2,
      alternative = "two.sided", var.equal = TRUE)
    # names(t.result)

    ## Collect the selected statistics in the result table
    result.table[i, "i"] <- i
    result.table[i, "statistic"] <- t.result["statistic"]
    result.table[i, "p.value"] <- t.result["p.value"]

    ## Compute some additional statistics about the samples
    result.table[i, "m1"] <- mean(x1) ## Mean of sample 1
    result.table[i, "m2"] <- mean(x2) ## Mean of sample 2
    result.table[i, "diff"] <- mean2 - mean1 ## Difference between sample means
  }

  return(result.table) ## This function returns the result table
}
```

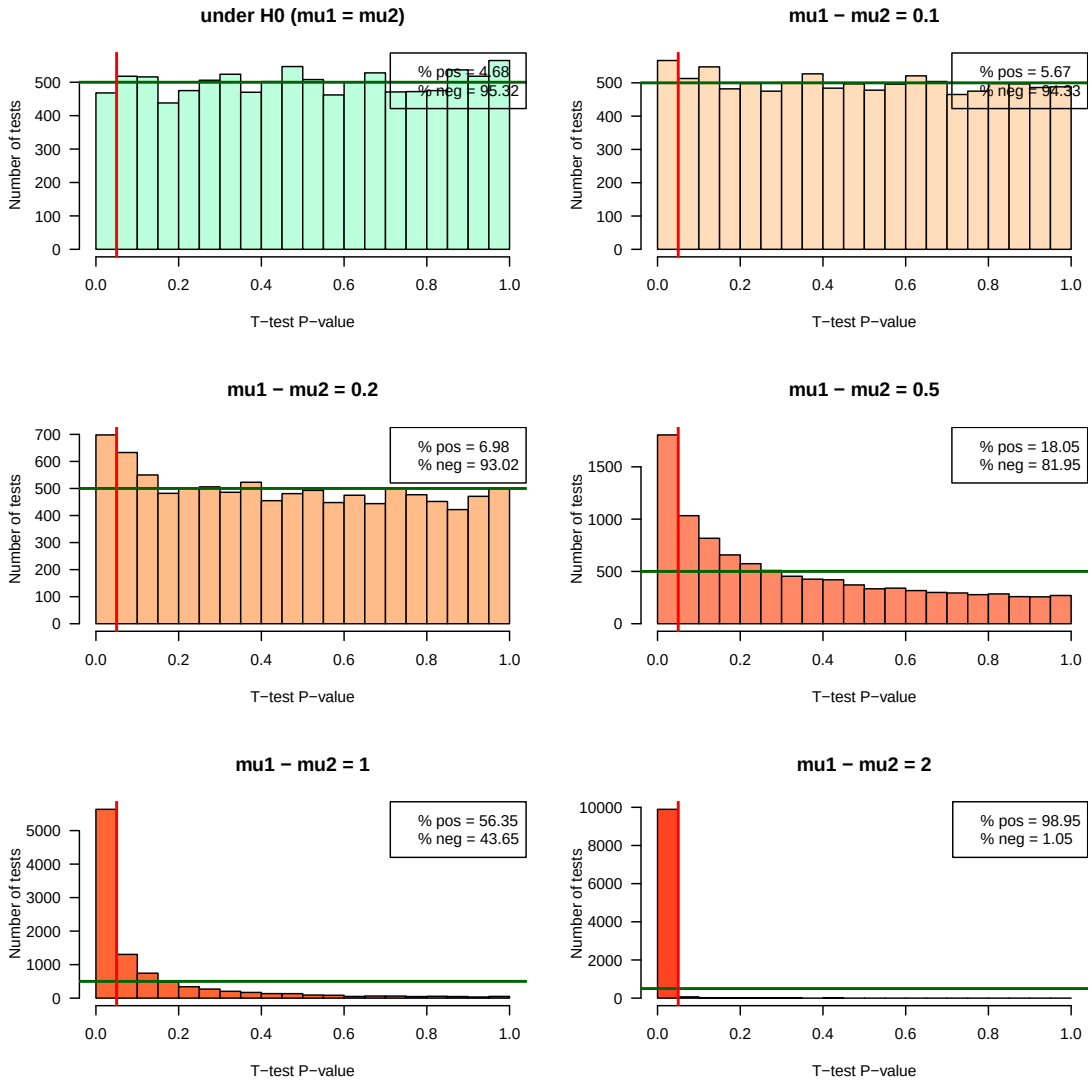
This function can then be used several times, with different values of the parameters.

```r
## What happens when  the two means are equal (under the null hypothesis)
d0 <- iterate.t.test(mu1 = 0, mu2 = 0)  ## Iterations

## Test increasing values of the difference between means
d0.1 <- iterate.t.test(mu1 = -0.05, mu2 = 0.05)  ## Iterations
d0.2 <- iterate.t.test(mu1 = -0.1, mu2 = 0.1)  ## Iterations
d0.5 <- iterate.t.test(mu1 = -0.25, mu2 = 0.25)  ## Iterations
d1 <- iterate.t.test(mu1 = -0.5, mu2 = 0.5)  ## Iterations
```

```r
d2 <- iterate.t.test(mu1 = -1, mu2 = 1)  ## Iterations
```

```r
## Define a function that rdraws the p-value histogram
## based on the result table of t-test iterations
## as produced by the iterate.t.test() function.
pvalHistogram <- function(
  result.table, ## required input (no default value): the result table from iterate.t.test()
  main = "P-value histogram",  ## main title (with default value)
  alpha = 0.05, ## Significance threshold
  ... ## Additional parameters, which will be passed to hist()
  ) {

  ## Plot the histogram
  hist(result.table$p.value,
       breaks = seq(from = 0, to = 1, by = 0.05),
       las = 1,
       xlim = c(0,1),
       main = main,
       xlab = "T-test P-value",
       ylab = "Number of tests", ...)

  ## Draw a horizontal line indicating the number of tests per bin that would be expected under null hyp
  abline(h = r / 20, col = "darkgreen", lwd = 2)
  abline(v = alpha, col = "red", lwd = 2)

  ## Compute the percent of positive and negative results``
  nb.pos <- sum(result.table$p.value < alpha)
  nb.neg <- r - nb.pos
  percent.pos <- 100 * nb.pos / r
  percent.neg <- 100 * nb.neg / r

  ## Add a legend indicating the percent of iterations declaed positive and negative, resp.
  legend("topright",
         legend = c(
           paste("% pos =", round(digits = 2, percent.pos)),
           paste("% neg =", round(digits = 2, percent.neg))
         ))
}
```

```r
par(mfrow = c(3, 2)) ## Prepare 2 x 2 panels figure
pvalHistogram(d0, main = "under H0 (mu1 = mu2)", col = "#BBFFDD")
pvalHistogram(d0.1, main = "mu1 - mu2 = 0.1", col = "#FFDDBB")
pvalHistogram(d0.2, main = "mu1 - mu2 = 0.2", col = "#FFBB88")
pvalHistogram(d0.5, main = "mu1 - mu2 = 0.5", col = "#FF8866")
pvalHistogram(d1, main = "mu1 - mu2 = 1", col = "#FF6633")
pvalHistogram(d2, main = "mu1 - mu2 = 2", col = "#FF4422")
```




```
par(mfrow = c(1, 1)) ## Restore single-panel layout for next figures
```