Statistical Inference for proportions

Alessandro Vasta

Dec 4th 2020

Overview

In this paper, I would like to bring to your attention an alternative approach to what is provided by the standard R library "stats" regarding statistical inference on proportion. Concerning the "stats" library I'll refer mainly the function binom.test() rather than prop.test() since I didn't notice a significant difference between the two, while I noticed a significant difference between the proposed approach and both of them. Moreover I took as probability model the binomial distribution.

Test_prop() function

test_prop() is one of the functions developed in this work. It is thought to provide interval limits expressed as maximum (n_up) and/or minumum (n_l) numbers of occurrences in order to accept or reject a null hypothesis represented by a proportion with the probability p_0. Of course those numbers depends on the number of trials (N) that have been performed. In addition this function take into consideration also an alternative proportion with probability p_1 to be related to a statistical power for a possible hypothesis test. So a conceptual initial difference with the R function binom.test() is that test_prop() is thought to determine intervals for hypothesis test rather than confidence intervals. Nevertheless binom.test() in next Monte Carlo simulations for comparison purposes, will be used in a compatible way assuming the ratio occurrences / trials matching p_0.

Data input list

- p_0 is the event probability correspondent to the null hypothesis Hp_0 of the test.
- p_1 is the event probability correspondent to the alternative hypothesis Hp_1 of the test.
- conf = 1 - α is the confidence of the test where α, the significance level, is the type I error, that is the probability to reject the null hypothesis when the null hypothesis is true ($p = p_0$).
- power = 1 - β, is the input power for the test used to calculate p_1@pow that is the event probability p_1 correspondent to selected input power. β is the Type II error, that is the probability to accept the null hypothesis when the real event probability is p_1@pow, so power is the minimum probability to detect a proportion $p \geq p_1$@pow if $p_1 > p_0$ or $p \leq p_1$@pow if $p_1 < p_0$.
- N is the number of trials.
- steps are the number of lines of the output table where all the outputs are calculated with different value for N. Each line refer a number of trials starting from the top = $N * 2^{i-1}$ if the parameter inc = -1, while N = N + inc*(i-1) if inc is an integer > 0, with i = 1,. . . ,steps = line index.
- inc [integer] is the incremental step to be performed according what specified in the above bullet point.
- side indicates the alternative hypothesis and must be one of "two", "greater" or "less"

Data output list

The output data provided in a data frame format for the above inputs are:

- N is the number of trials.
- n_l is the minimum number of events over N trials that shall be collected without rejecting the null hypothesis. If this limit will be exceeded the null hypothesis ($p = p_0$) shall be rejected.

- n_up is the maximum number of events over N trials that can be collected without rejecting the null hypothesis. If this limit will be exceeded the null hypothesis ($p = p_0$) shall be rejected.
- pr_l(%) is the percentage proportion correspondent to the proportion n_l/N
- pr_up(%) is the percentage proportion correspondent to the proportion n_up/N
- conf(%) is the real output confidence of the test that is different from what specified as input because of the quantization error of the number of events (n) that are constrained to be integers.
- pwr%@p1= ... % is the effective statistical power of the test calculated with an effect size equal to the input p_1. This could be a small value for the lowest values of N. (see below the data tables).
- p1%@pwr ... % is the effect size p_1 to allow the statistical power specified in input data.

So, this is the R code for the function:

```
test_prop <- function(conf,N,p0,p1,power,steps,inc = -1,side = "less") {
    conf_int <- data.frame(N = 0, n_l = NA, n_up = NA, prop_l = NA, prop_up = NA,
                           conf = 0, power = 0, p1.pow = 0)
    names(conf_int)[2] <- " n_l"
    names(conf_int)[3] <- " n_up"
    names(conf_int)[4] <- " pr_l(%)"
    names(conf_int)[5] <- " pr_up(%)"
    names(conf_int)[6] <- " conf(%)"
    names(conf_int)[7] <- paste(" pwr%@p1=",as.character(p1*100),"%")
    names(conf_int)[8] <- paste(" p1%@pwr",as.character(power*100),"%")
    alpha <- 1 - conf
    conf2 <- conf
    accuracy <- 26
    if (side == "two") {
        conf <- 1 - alpha/2
        side_l <- c(T,T)
    } else if(side == "less") side_l <- c(F,T)
    else side_l <- c(T,F)

    if (side_l[1] & side_l[2]) conf <- 1 - alpha/2
    for (i in 1:steps) {
        if(inc == -1) {
            Nx <- N*2^(i-1)
        } else Nx <- N + inc*(i - 1)

        # n_l is the minimum number of events that does not reject Hp0 (p0)
        # n_up is the maximum number of events that does not reject Hp0 (p0)
        if(side_l[1]) {
            n_l <- qbinom(conf,Nx,p0,lower.tail = FALSE)
            if(n_l == 0) stop("(*1) Low side test not possible:
                              Low Number of trials")
            alpha2 <- pbinom(n_l,Nx,p0)
            if(side_l[2]) n_l <- n_l + 1
            conf2 <- 1 - (alpha -alpha2)
            if (conf2 >= 1) conf2 <- (1 - alpha/100)
        } else n_l <- NA
        if(side_l[2]) n_up <- qbinom(conf2,Nx,p0) else n_up <- NA
        # prop_xx are the min/max proportion allowed
        if(side_l[1]) prop_l <- round(((n_l)/Nx)*100,digits = 4) else prop_l <- NA
        if(side_l[2]) prop_up <- round((n_up/Nx)*100,digits = 4) else prop_up <- NA
        # conf_r is the real confidence after the implementation of the test
        if (side_l[1] & side_l[2]) {
            conf_r <- round((pbinom(n_up,Nx,p0) - pbinom(n_l-1,Nx,p0))*100,
```

```
                            digits = 2)
        } else if (side_l[2]) {
            conf_r <- round(pbinom(n_up,Nx,p0)*100,digits = 2)
        } else {
            conf_r <- round((1-pbinom(n_l-1,Nx,p0))*100, digits = 2)
        }
        # This condition is satisfied when a Low side detection is not possible
        if (conf_r/((1-alpha)*100) < 0.9) {
            stop("(*2) Low side test not possible: Low Number of trials")
        }
        # real test power at input p1
        if(side_l[2] && (p1 > p0)) {
            power_r <- round(100*(1-pbinom(n_up,Nx,p1)),digits = 2)
        } else if (side_l[1] && (p1 < p0)) {
            power_r <- round(100*(pbinom(n_l-1,Nx,p1)),digits = 2)
        } else stop("side and p1 are not consistent")

        # calculation of p1 correspondent to the input power
        p1_pow <- 0.5
        p1_inc <- 0.25
        iter_for_accuracy <- 0
        while (iter_for_accuracy < accuracy) {
            iter_for_accuracy <- iter_for_accuracy + 1
            if(p1 > p0) {
                powerx <- (1 - pbinom(n_up,Nx,p1_pow))
            } else {
                powerx <- pbinom(n_l-1,Nx,p1_pow)
            }

            if (powerx > power) {
                p1_pow <- p1_pow - p1_inc*sign(p1-p0)
            } else {
                p1_pow <- p1_pow + p1_inc*sign(p1-p0)
            }
            p1_inc <- p1_inc / 2
        }
        p1_pow <- round(p1_pow*100, digits = 4)
        conf_int[i,] <- c(Nx, n_l,n_up, prop_l,prop_up, conf_r, power_r, p1_pow)
    }
    conf_int
}
```

Data out examples

In this first case we'll see as the option inc = -1 could provide a way to gain some level of awareness regarding the approximated sample size N needed to achieve an expected power level (pwr%@p1). Moreover, it is possible to see the extent of the alternative hypothesis p1 to be detected with the desired input power (p1(%)@pwr).

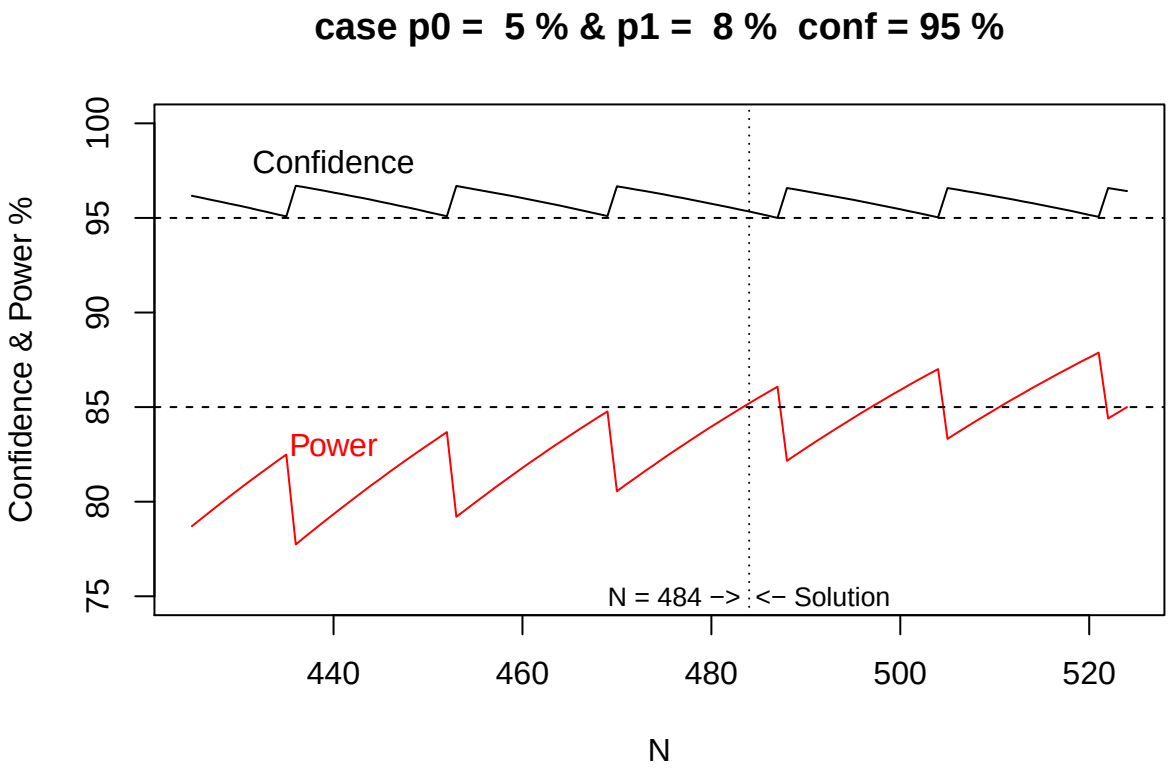
```
N <- 50
inc <- -1
steps <- 12
conf <- 0.95
power <- 0.95
p0 <- 0.0005
p1 <- 0.001
side <- "less"

test_prop(conf,N,p0,p1,power,steps,inc,side)
```

```
        N  n_l  n_up  pr_l(%)  pr_up(%)  conf(%)  pwr%@p1= 0.1 %  p1%@pwr 95 %
1      50   NA     0       NA    0.0000    97.53            4.88        5.8155
2     100   NA     0       NA    0.0000    95.12            9.52        2.9513
3     200   NA     1       NA    0.5000    99.53            1.75        2.3498
4     400   NA     1       NA    0.2500    98.25            6.15        1.1804
5     800   NA     2       NA    0.2500    99.21            4.73        0.7849
6    1600   NA     2       NA    0.1250    95.26           21.66        0.3930
7    3200   NA     4       NA    0.1250    97.64           21.93        0.2858
8    6400   NA     6       NA    0.0938    95.54           45.77        0.1850
9   12800   NA    11       NA    0.0859    96.93           62.63        0.1422
10  25600   NA    19       NA    0.0742    96.26           88.97        0.1089
11  51200   NA    34       NA    0.0664    95.56           99.30        0.0884
12 102400   NA    63       NA    0.0615    95.34          100.00        0.0759
```

Another way to use this function is with constant increment, may be when is clear approximately the value of N. In this case, a plot of confidence and a power trends could be more effective. Let's consider the following inputs:

```
N <- 425
steps <- 100
conf <- 0.95
p0 <- 0.05
p1 <- 0.08
power <- 0.85
side <- "less"
inc <- 1
```

And this is the output



The steps in the above saw-tooth lines are related to the change of n_up to the next integer value. So, confidence and power are far to be continuous and monotonic and the analysis of this trend has been the conceptual base for the implementation of the next function, that is: the determination of the minimum number of trials (the sample size), given confidence, power, p_0 and p_1.

The ss_prop() function

Referring the previous plot we can see that a solution matching a specific value for power and confidence, like in the example with power = 85% and confidence = 95%, does not exist. Nevertheless with the approximation forced by integers quantization we could find a solution that best approximate the power and confidence goals. It's worth noting that this potential solution is not unique, in fact, identifying these solutions with the intersection of power with the horizontal line 85%, for the reported example, there are 7 possible solutions fulfilling this choice. Obviously what we are looking for is the one with the lower N. This task is achieved by the following function starting with the a low value for N initialized at 70%, the solution provided by the normal approximation. The first part of the algorithm make N increasing with incremental values such that all iteration step could match a power spike. This first phase end as soon as the first power spike cross the power goal. If the current pre spike segment is increasing, a bisection method based algorithm move N backward up to the power goal crossing. This approach achieves the lowest N satisfying the constraints: confidence $\geq$ confidence set and power $\geq$ power set.

this is the R code for the function:

```
# function for sample size determination
ss_prop <- function(p0,p1,conf,power,side) {
    if(conf < 0.6) stop("Confidence too low (< 60%) ")
    if(power < 0.6) stop("Power too low (< 60%) ")
    alpha <- 1 - conf
    beta <- 1 - power
    if (side == "two") alpha <- alpha/2
    Max_iter <- 10000
    n_iter <- 0
```

```
    # Nr = sample size aproximation based on normal distribution
    Nr <- ((qnorm(alpha,lower.tail = F)*sqrt(p0*(1-p0))+
                qnorm(beta,lower.tail = F)*sqrt(p1*(1-p1)))/abs(p1-p0))^2
    # N => first sample size initialization for next iterations
    # This is a starting point with a power always lower than the power set
    N <- round(Nr*0.7,digits = 0)
    k <- round(log2(Nr*0.4),digits =0)

    # first iteration loop by increasing N values with step calculated by peak_det_xx
    pwr <- pwr_calc(conf,N,p0,p1,power,side)
    while ((pwr[1,7] < power*100) & (n_iter < Max_iter)) {
        if(side == "less") {
            N <- peak_det_p0p1(N,conf,p0,k)
        } else {N <- peak_det_p1p0(N,conf,p0,k)}
        pwr <- pwr_calc(conf,N,p0,p1,power,side)
        n_iter <- n_iter + 1
    }
    if (n_iter >= Max_iter) stop("Algorithm not convergent: p0 & p1 too close")

    # 2nd iteration loop: Active only in a local interval when power(N) is increasing
    # This loop find the minimum N satisfying confidence > set & power > set
    k <- round(log2(N*0.3),digits =0)
    dN <- 2^k
    iter_for_k <- 0
    while (iter_for_k <=k) {
        if (pwr[1,7] > power*100) {
            N <- N - dN
        } else N <- N + dN
        pwr <- pwr_calc(conf,N,p0,p1,power,side)
        iter_for_k <- iter_for_k +1
        dN <- dN / 2
    }
    if(pwr[1,7] < power*100) N <- N + 1
    pwr <- pwr_calc(conf,N,p0,p1,power,side)

    if(pwr[1,7] > power*120 ) stop("Unsuitable parameters setting")
    pwr
}
```

and the sample size calculation for the last example, where $p_0 = 5\%$ and $p_1 = 8\%$, confidence $= 95\%$ and power $= 85\%$ is given by:

```
ss_prop(p0,p1,conf,power,side)
```

```
    N  n_l  n_up  pr_l(%)  pr_up(%)  conf(%) power(%) side
1 484   NA    32       NA    6.6116    95.34     85.2 less
```

Note: the input and output data list have the same meaning described for function test_prop(). Furthermore, this function provides in a single step, the sample size N and the test limits n_l and n_up to implement the test with the required input p_0, p_1, confidence and power.

Monte Carlo simulations and comparison with R functions

The following tables show the simulations with the binomial distribution function rbinom() performed with 1,000,000 observations for each presented case. In the tables below the meanings of the output data is given

by the following notes:

- N is the number of trials simulated 1 million times, that is with the function rbinom(1e6,N,p0) for Hp_0 data stream (to verify the confidence) and rbinom(1e6,N,p1) for Hp_1 data stream (to verify the power).
- p0(%) is the percentage probability of event occurrence for Hp_0 simulation function, (as above mentioned) and, at the same time, the input for test_prop() and binom.test() for the calculation of interval acceptance limits n_l, n_up.
- p1(%) is the percentage probability of event occurrence for Hp_1 simulation function, (as above mentioned) and, at the same time, the input for ss_prop() and power.prop.test() to determine the sample size N. p1(%), in any case, is always used in simulation to calculate the effective test power, even in all cases where N is not determined by power considerations.
- co(%) is the percentage probability of confidence defined as input set for the test.
- coP(%) is the percentage probability of confidence calculated as best fit by the function test_prop() (my proposed function) that, as mentioned before can't be equal to co(%) because n_l and/or n_up are forced to be integer.
- co_sP(%) is the actual percentage probability of confidence resulting from the simulation, with the test limits calculated by the proposed functions test_prop() or ss_prop().
- pwr_sP() is the actual percentage probability of power resulting from the simulation, for the specified value p1(%), with the test limits calculated by the proposed functions test_prop() or ss_prop(). (In this case the simulated data stream is generated by rbinom(1e6,N,p1))
- co_sR(%) is the actual percentage probability of confidence resulting from the simulation, with the test limits calculated by the R function binom.test().
- pwr_sP() is the actual percentage probability of power resulting from the simulation, for the specified value p1(%), with the test limits calculated by the R function binom.test().
- side is the alternative hypothesis indicator

Performance evaluation of function test_prop()

Finally the table below shows the behavior of function test_prop() for some cases where the the sample size N has been freely set without any particular consideration about power, nevertheless the power performances are still evaluated respect the p1(%) input.

	N	p0(%)	p1(%)	co(%)	co_P(%)	co_sP(%)	pwr_sP(%)	side
1	250	1	3	95	95.88	95.9	76.3	less
2	500	1	3	85	86.77	86.8	98.3	less
3	1200	10	11	90	90.16	90.1	44.1	less
4	15000	0.05	0.1	99	99.54	99.5	43.1	less
5	5000	1	0.6	95	95.34	95.4	93.5	greater
6	5000	0.1	0.01	95	95.96	96	91	greater
7	800	10	14.2	95	95.24	95.3	93.9	two
8	800	10	6.47	95	95.24	95.2	96.3	two
9	124	0.5	5	95	97.52	97.5	95	less

From the above table we can notice that the experimental confidence co_sP(%) always matches the calculated value co_P(%) by test_prop().

Comparison of function test_prop() with binom.test()

The following table refers the same cases as the previous one:

	N	p0(%)	p1(%)	co(%)	co_sP(%)	co_sR(%)	pwr_sP(%)	pwr_sR(%)	side
1	250	1	3	95	95.9	98.6	76.3	62.6	less
2	500	1	3	85	86.8	93.3	98.3	96.5	less
3	1200	10	11	90	90.1	91.7	44.1	40.5	less
4	15000	0.05	0.1	99	99.5	99.9	43.1	25.1	less
5	5000	1	0.6	95	95.4	93.7	93.5	95.4	greater
6	5000	0.1	0.01	95	96	96	91	91	greater
7	800	10	14.2	95	95.3	95.3	93.9	93.9	two
8	800	10	6.47	95	95.2	95.2	96.3	96.3	two
9	124	0.5	5	95	97.5	100	95	74.9	less

The R function binom.test() seems more inaccurate achieving the confidence set. For one sided interval, when the side is "less", the actual confidence always exceed the better approximation provided by test_prop(). The consequence is a worse performance concerning power for the same value of p1. For one side "greater" intervals, in one case binom.test() fails achieving the confidence input set, while for two sided intervals we have the same behavior. However, these differences are still not as impressive as can be achieved by comparing ss_prop() and power.prop.test().

Performance evaluation of function ss_prop()

The following table shows the best fit for the input parameters p0(%),p1(%),co(%) and pwr(%) for the proposed function ss_prop(), with the simulation results always obtained with 1 million observations for each case.

	N	p0(%)	p1(%)	co(%)	pwr(%)	co_P(%)	co_sP(%)	pwr_sP(%)	side
1	362	0.5	2	95	85	96.33	96.3	85.1	less
3	484	5	8	95	85	95.34	95.3	85.2	less
5	694	5	3	95	85	95.06	95	85	greater
7	673	5	8	95	85	95.43	95.4	85.2	two

Comparison of function s_prop() with power.prop.test()

Now let's compare ss_prop() and power.prop.test() output results for the same list of input parameters corresponding to the first line of the above table

```
p0 <- 0.005
p1 <- 0.02
conf <- 0.95
power <- 0.85
side <- "less"

dat <- ss_prop(p0,p1,conf,power,side)
dat
```

	N	n_l	n_up	pr_l(%)	pr_up(%)	conf(%)	power(%)	side
1	362	NA	4	NA	1.105	96.33	85.05	less

while the corresponding R function gives:

```
power.prop.test(p1=p0,p2=p1,sig.level = 1-conf,power = power,
                alternative = "one")
```

```
     Two-sample comparison of proportions power calculation

              n = 787.4355
             p1 = 0.005
             p2 = 0.02
      sig.level = 0.05
          power = 0.85
    alternative = one.sided

NOTE: n is number in *each* group
```

The main difference we can see from comparing the output is that the sample size calculated by power.prop.test() is more than twice that calculated by ss_prop() despite the solution of ss_prop(), (directly available by n_up test limit), has been already positively verified by simulations. An additional difference is about the confidence and power output that in ss_prop() are the actual value implemented by the test rather than the theoretical input values. The next table shows the comparison of the two functions for all cases of the previous table. In this simulation the test limits by R functions are obtained again with binom.test().

	N	p0(%)	p1(%)	co(%)	pwr(%)	co_sP(%)	co_sR(%)	pwr_sP(%)	pwr_sR(%)	side
1	362	0.5	2	95	85	96.3	99.7	85.1	58.8	less
3	484	5	8	95	85	95.3	96.9	85.2	80.8	less
5	694	5	3	95	85	95	92.8	85	89.3	greater
7	673	5	8	95	85	95.4	95.9	85.2	81.5	two

The simulation shows that the test limits (not shown here) calculated by binom.test(), never satisfy the input data as minimum requirements. Especially when side = "less" or "two" the actual power is always lower and sometimes much lower than the power input and vice versa, when side = "greater" is the actual confidence to be lower than the expected input.

The next table shows instead the same comparisons schema, but performed with the sample sizes calculated by function power.prop.test(), always referring the same cases.

	N	p0(%)	p1(%)	co(%)	pwr(%)	co_sP(%)	co_sR(%)	pwr_sP(%)	pwr_sR(%)	side
2	787	0.5	2	95	85	95.3	99.3	98.9	95.2	less
4	970	5	8	95	85	95.8	95.8	98.1	98.1	less
6	1379	5	3	95	85	95.5	94.1	98.4	98.9	greater
8	1211	5	8	95	85	95.1	95.7	98.7	98.2	two

The sample sizes N calculated by power.prop.test() are always roughly **double those calculated by ss_prop(),** nevertheless in case at line 6, corresponding to line 5 on the previous table and with side = "greater", the actual confidence is still lower than the expected input.

Conclusions

Dear all, I started this work as a personal exercise to become familiar with with R and Rstudio, after that it turned out step by step in a real research. The ability to determine a sample size of 50% or even less than what is currently calculated for the same input requirements seems to me a significant result. I run this code and developed this Rmarkdown on R version 4.0.3 (2020-10-10) – "Bunny-Wunnies Freak Out", but I also tested it on the older version 3.4. You can find all R code in the attached Rmd file. I may be the victim of an oversight, if this is the case please give me feedback. If not, please give me feedback anyway :)