Computing exact-test parameters for binomial MaxSPRT

Brock Stewart, Ph.D.[†] and Rongxia Li, Ph.D.
CDC/DHQP Immunization Safety Office

CDC Statistics Day 2014 – August 27

1. Objectives

The Maximized Sequential Probability Ratio Test (MaxSPRT) is frequently used in post-marketing vaccine safety surveillance. Our goal is to develop algorithms to quickly compute exact-test

- critical value
- Type I error
- power
- data points that suggest reject H_0
- data points that suggest accept H_0

for binomial MaxSPRT, and to produce software in the form of R and C source codes.

2. Binomial MaxSPRT

Data: sequence of *iid* Bernoulli(p) trials
Comparison: $H_0 : p = p_0$ vs. $H_1 : p > p_0$
Repeat: With s successes and f failures following $n = s + f$ trials compute the log-likelihood ratio

$$LLR(p_0|s, f) = \log\left(\frac{Bin(s|n, \hat{p})}{Bin(s|n, p_0)}\right)$$

where $\hat{p} = s/n$; $LLR = 0$ unless $s/p_0 > f/(1 - p_0)$
Until: $LLR(p_0|s, f) \geq V$ (reject H_0) or $n = N$
where V and N are predetermined limits

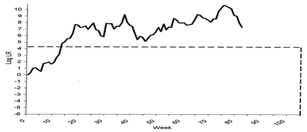

Figure 1: Example LLR vs. time (solid), with critical value and upper time limit (dashed).

3. Example Applications

The MaxSPRT method has been used in at least 8 post-marketing vaccine safety studies since 2007. For example, Tse et al. (2014) conducted a study that involved monitoring data from nine CDC Vaccine Safety Datalink managed care organization sites; it was noted whether International Classification of Disease (ICD) codes for seizures, Guillain-Barre syndrome, encephalitis, and anaphylaxis occurred within one of two predefined time windows following receipt of inactivated influenza vaccine (IIV) or live attenuated influenza vaccine (LAIV): a *risk interval* immediately following the vaccination date; and a *control interval* beyond the risk interval at a time not likely to be biologically plausible or causally associated with the vaccine. In this design, each vaccinee represents a Bernoulli trial, with "success" defined as ICD code occurring in the risk interval.

4. Notation

Rejection points: suggest reject H_0 if observed:

$$\mathcal{R}_{V,N,p_0} = \{(s, f) \mid LLR(p_0|s, f) \geq V \text{ and } s + f \leq N\}$$

Observable points: some (s, f) points may not be observable since analysis stops when $LLR(p_0|s, f) \geq V$

$$\mathcal{O}_{V,N,p_0} = \{(s, f) \text{ that are observable and } s + f \leq N\}$$

Stopping points: Observable rejection points:

$$\mathcal{X}_{V,N,p_0} = \mathcal{R}_{V,N,p_0} \cap \mathcal{O}_{V,N,p_0}$$

5. Methodological Setup

Before beginning a binomial MaxSPRT analysis the researchers choose values for p_0, N, and V. The value of V is chosen to reflect the desired Type I error rate α. Kulldorff et al. (2011) provides a table of critical values corresponding to an exact test for several values of N, p_0, and α. The exact value of V for other N, p_0, and/or α must be found through an iterative numerical routine. Specifically, for given N and p_0, Type I error is a decreasing step function of V, and our motivation is to develop an algorithm to quickly find a value of V that gives a desired Type I error.

Our interest is in computing the probability of rejecting H_0 for a given set of stopping points $\mathcal{X}$ and success probability p, which is

$$\beta(p, \mathcal{X}) \equiv Pr(\text{reject } H_0|p, \mathcal{X}) = \sum_{x \in \mathcal{X}} Pr(x|p, \mathcal{X}),$$

where each $x \in \mathcal{X}$ is an (s, f)-point. Type I error is given by $\beta(p_0, \mathcal{X}_{V,N,p_0})$, and power is $\beta(p_1, \mathcal{X}_{V,N,p_0})$ for $p_1 > p_0$. For given N and p_0, a value of V that gives a desired Type I error, α, can be found by solving

$$\min_V |\beta(p_0, \mathcal{X}_{V,N,p_0}) - \alpha|.$$

Note that any routine to perform this minimization requires – for each iteration – determining the set $\mathcal{X}_{V,N,p_0}$, and for each $x \in \mathcal{X}_{V,N,p_0}$ computing $Pr(x|p_0, \mathcal{X}_{V,N,p_0})$, which are generally not pure binomial probabilities since observing some (s, f)-points result in stopping the analysis.

Here, we present two algorithms to quickly compute the set $\mathcal{X}_{V,N,p_0}$ and to compute the exact probability of rejecting the null hypothesis in a binomial MaxSPRT analysis for given values of V, N, p_0, and any given true success probability p. This is useful for computing exact critical values, Type I error, power, and other ancillary information. We call the two algorithms the "Tree" and "Lattice" algorithms based on the heuristic devices we use to conceptualize the algorithms.

6. Tree Algorithm

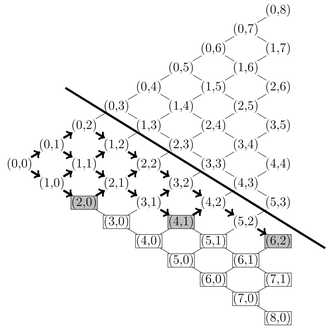

Figure 2: An (s, f) Tree for $N = 8$; also, $V = 0.75$, $p_0 = 0.5$ give: acceptance boundary (bold line); rejection points (boxed); stopping points (shaded box). Each point with an arrow pointing to it is evaluated by the Tree Algorithm.

With $P_{s,f} \equiv Pr(s, f|p)$ and $P_{0,0} = 1$ we have

$$P_{s,f} = p \times P_{s-1,f} + (1 - p) \times P_{s,f-1} \times I_{(s,f-1) \notin \mathcal{R}}$$

This of course leads to $P_{s,f} = Bin(s|n, p)$ if all points are observable. If some points are not observable, i.e. if there are stopping points, then the $P_{s,f}$ are no longer binomial probabilities for most points. The procedure is to

- **Loop:** over columns (a vertical collection of points in the tree, e.g. $(0, 2)$, $(1, 1)$, and (2.0))
- **Loop:** over points within each column, computing probabilities using the above formula
- **Exact Type I error:** when a stopping point is encountered, add it's probability to a cumulative sum representing $\beta(p_0, \mathcal{X}_{V,N,p_0})$
- **shrink from below**: do not evaluate points that can only be reached by first going through a stopping point, i.e. un-observable points
- **shrink from above**: do not evaluate points above the acceptance boundary

Previous work had utilized essentially the same procedure sans shrinking from above/below. Our algorithm substantially reduces the search space and thus run time. Note: $f^* = \max\{f \mid (N - f, f) \in \mathcal{R}\}$ is the largest number of failures in N trials that result in rejecting H_0.

```
input V, N, p0
β ← 0
Compute f*
π^(1) ← (1 − p, p)                    ▷ Initialize for n = 1
s_a^(1) ← 0,
s_b^(1) ← 1
if LLR(p0|1, 0) ≥ V then
    β ← β + (last element of π^(1))
end if
for n in 2 . . . N do                 ▷ Loop over remainder of columns
    s_a^(n) ← max(0, n − f*)          ▷ shrink from above
    s_b^(n) ← s_a^(n−1) + 1
    if LLR(p0|s_b^(n−1), n − s_b^(n−1)) ≥ V then
        s_b^(n) ← s_b^(n−1)           ▷ shrink from below
        π^(n−1) ← π^(n)[1 : (s_b^(n−1) − s_a^(n−1))]
    end if                            ▷ compute probs
    π^(n) ← p × P_{s−1,f} + (1 − p) × P_{s,f−1} × I_{(s,f−1)∉R}
    if LLR(p0|s_b^(n), n − s_b^(n)) ≥ V then
        β ← β + (last element of π^(n))
    end if
    if LLR(p0|s_b^(n), n − s_b^(n)) ≥ V AND s_a^(n) = s_b^(n) then
        Exit and return β
    end if
end for
return β
```

7. Lattice Algorithm

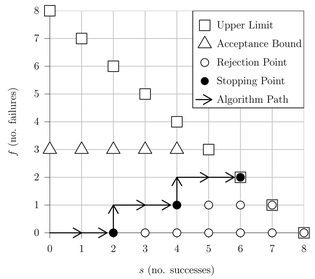

Figure 3: Lattice conceptualization for $N = 8$; also, $V = 0.75$, $p_0 = 0.5$ give: acceptance boundary (triangles); rejection points (circles); stopping points (filled circles). Arrows indicate path taken by the Lattice Algorithm.

Finding Stopping Points: It can be shown that the monotonicity of $LLR(p_0|s, f)$ implies: (i) there is at most one stopping point in each lattice row; (ii) if both (s_1, f) and $(s_2, f + 1)$ are stopping points, then $s_1 \leq s_2$. Thus: (a) the algorithm travels through the lattice in a step pattern; (b) evaluates only N points; (c) the set of stopping points can be represented as $\mathcal{X} = \{(s_i, i)\}_{i=0,\ldots,f^*}$ with the property that $s_i \leq s_{i+1} \forall i < f^*$, or more succinctly as a list $\mathcal{X}' = \{s_i\}_{i=0,\ldots,f^*}$ of increasing s-values, where s_i is the s-value of the point (s_i, i).

Computing Probabilities: Let $x_i = (s_i, i)$. Clearly, if $\mathcal{R} = \emptyset$ then $Pr(s, f) = Bin(s|n, p)$, because there are $\binom{n}{s}$ paths from $(0, 0)$ to (s, f), each having probability $p^s(1 - p)^f$. The probability of observing (s, f) given that some paths are blocked by stopping points can be obtained by replacing $\binom{n}{s}$ with the number of paths not passing through stopping points, $T(x|\mathcal{X})$

$$Pr(x|p, \mathcal{X}) = T(x|\mathcal{X})p^s(1 - p)^f.$$

The no. of un-blocked paths to x_i can be computed as

$$T(x_i|\mathcal{X}) = \binom{n_i}{s_i} - \sum_{j<i} T(x_j|\mathcal{X}) \times \binom{n_{ij}}{s_{ij}},$$

where $n_i = s_i + i$, $n_{ij} = s_i + i - s_j - j$, and $s_{ij} = s_i - s_j$. The second term represents the number of paths from $(0, 0)$ to x_i that first pass through a stopping point to the left of x_i. Combining these equations we obtain

$$\pi_i \equiv Pr(x_i|p, \mathcal{X}) = Bin(s_i|n_i, p) - \sum_{j<i} \pi_j \times Bin(s_{ij}|n_{ij}, p).$$

```
function ProbRejectH0(V, N, p0, p1)
    X', X ← ∅
    s, f, β ← 0
    while s + f ≤ N do
        if LLR(p0|s, f) ≥ V then
            x ← (s, f)
            X ← (X, x)
            π ← Pr(x|p1, X, P)
            P ← (P, π)
            β ← β + π
            X' ← {X', s}
            f ← f + 1
        else
            s ← s + 1
        end if
    end while
return X', β
end function
```

References

Kulldorff et al. (2011) Sequential Analysis 30(1):58-78 | Tse et al. (2014) Pharmacoepi. & Drug Safety 23(5):548-53

Contact

[†] jnn6@cdc.gov | (404) 639-0215 | NCHHSTP/DTBE

8. Additional Information

These two algorithms compute α for given V; an iterative routine can thus be used to perform the minimization indicated in Section 5 to obtain an exact-test V for given α. Relatively minor modifications to these algorithms additionally allows for computation of: power; lists of (s, f) points that suggest rejecting and accepting H_0, respectivley, if observed.

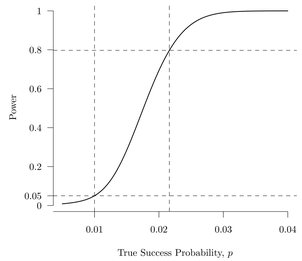

Figure 4: An example power curve

9. Run Times

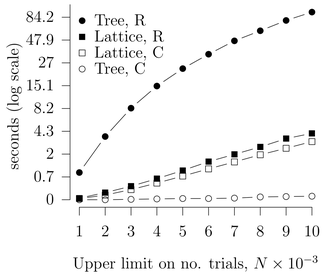

Figure 5: Run times for upper limits of $N = 1 \times 10^3$ – 1×10^4.

The Tree algorithm produced both the fastest and slowest run times, depending on whether it was run as an R or C function; this is likely due to the nested loops required. On the other hand, the Lattice Algorithm had similar run times between the R and C implementations, which is likely due to the lack of nested loops and the need to compute binomial coefficients (the R function to do this calls a C subroutine).