TBSS Poisson Power Calculation

Catherine Mahoney

Power calculation for tree-based scan statistics with a Poisson probability model

This pipeline automates the calculation of power for tree-based scan statistics used in pharmacovigilance using the recently published, open-source implementation in the package `TBSS`. This version uses a Poisson probability model. To evaluate the power of the tree-based scan statistic, it implements a plasmode simulation procedure to introduce known elevation in relative risk, as well as a generic stratification procedure for the estimation of background rates, then calculate the proportion of trials in which the simulated node alerts. The pipeline uses the `targets` and `crew` packages to manage the parallel simulations and subsequent calculations.

Input data

The specification of the precise study design, including inclusion/exclusion criteria, baseline period, and observation period are out of the scope of this pipeline. It is assumed that the user has created the cohorts according to their selected design and formatted the table as below, with a column for *id*, *treatment_group*, each relevant covariate, and a single *outcome* column with one row for each patient-outcome combination.

This vignette uses a simulated data set of treatment group, covariates, and outcomes from a subset of ICD-10-CM diagnosis codes corresponding to diseases of the respiratory system (J00-J99). The subset of codes is used for demonstration purposes to reduce run time.

Sample input data

id	treatment_group	cov_1	cov_2	...	outcome
1	1	0	1	...	A047
1	1	0	1	...	J459
1	1	0	1	...	E119
2	0	1	0	...	M545
2	0	1	0	...	F321

Tree-structured variable

Tree-based scan statistics leverage the hierarchical nature of coding systems such as ICD-10-CM to test for excess risk as multiple resolutions. TBSS requires that the tree be formatted with a *parent* and *child* column, as shown below:

child	parent
J040	J04
J041	J04
J0410	J041
J0411	J041

The function `tree_builder()`, contained in the repository, accepts a vector of codes and generates an appropriately formatted tree-structured variable.

Pipeline steps

The example pipeline is detailed in the _targets.R file, and each target and corresponding function or input contains comments to guide users in incorporating their own data and study design choices. Briefly, the steps in each simulation are as follows:

1. `select_subsample()` Select treatment and comparator groups of the desired size; this example selects 1,000 treatment and 1,000 comparator patients
2. `plasmode()` Introduce known elevation in relative risk of an outcome using plasmode simulation. This example uses a relative risk of 2. The user can decide whether to select a specific outcome in which to simulate excess risk or to specify a target incidence rate for the outcome to enrich and allow the function to selcet the outcome randomly. In this instance, a target incidence of .01 is specified.
3. `ps_strat()` Stratify the data based on a user-defined propensity score function and number of strata as defined in `strat_params.R` to calculate expected outcome rates. In this example, the propensity score is computed as a logistic regression using sample baseline data corresponding to categories from the Elixhauser comorbidity index, but users should customise to their own study and may choose to introduce potential confounding through choice of propensity score parameters. There are ten strata in this example but this parameter is also customisable.
4. `tbss_mask()` Apply TBSS to determine if the simulated elevation in risk is detected while preventing detection of unrelated statistical alerts.

This process is repeated many times to determine the power, calculated as the proportion of simulations wherein the simulated excess risk is detected. The example uses only 10 simulations for demonstration purposes; suitable numbers will vary by application, but 1,000 may be a good reference point. The package `crew` manages parallelisation. In this example, only two workers are used, but the user can customise to their own computer system.

Modifying `strat_params.R`

To specify the parameters for the stratification and estimation of expected outcome rates, the user must modify the parameters in the `strat_params.R` script.

The sample arguments correspond to a binomial GLM with a logit link function, but the user can specify any valid arguments to `glm()` in the `strat_args` list, along with the desired number of strata.

Contents of the `_targets.R` script

The script `_targets.R` directs the flow of work in the pipeline. Once the stratification parameters in `strat_params.R` have been specified, lines with a comment can be customised to required inputs and file locations for input and tree data.

```
library(targets)
library(crew)

tar_source() #configure metadata with matching parameters
tar_option_set(packages = c("readr", "dplyr", "tidyr", "TBSS"),
               controller = crew_controller_local(workers = 2) #specify workers
)

list(
  tar_target(file,
             "data/test_data_j.csv",    #location of your outcome data
             format = "file"
  ),
  tar_target(data,
             get_data(file)
  ),
  tar_target(iteration_id,
             1:10                        #specify number of replicates
  ),
  tar_target(subsample,
             select_subsample(data, 1000, 1000), #specify sample sizes
```

```
            pattern = map(iteration_id)
  ),
  tar_target(sim_data,
            plasmode(subsample,
                     target_outcome = NULL, #specify either outcome or incidence
                     target_inc = .01,
                     rr=2),                 #specify relative risk
            pattern = map(subsample)
  ),
  tar_target(ps_data,
            ps_strat(sim_data,
                     strat_args),
            pattern = map(sim_data)
  ),
  tar_target(tree_file,
            'data/test_tree.csv',                   #include tree file location
            format = "file"
  ),
  tar_target(tree,
            get_data(tree_file)
  ),
  tar_target(tbss,
            tbss_mask(ps_data,
                      model = "poisson",
                      tree),
            pattern = map(ps_data)
  ),
  tar_target(final_vector,
            unlist(tbss)
  ),
  tar_target(final_power,
            print(mean(final_vector))
  )

)
```

Running the pipeline

Run `tar_make()` to launch the pipeline. The output will enumerate each target and show which branches were already up-to-date. Finally, `tar_read(final_power)` will display the result of the complete power calculation. Here, TBSS had 20% power to detect an outcome

with an incidence of .01 and a relative risk of 2.

```
tar_make()
```

```
- The project is out-of-sync -- use `renv::status()` for details.
  start target iteration_id
  start target tree_file
  built target iteration_id [0.22 seconds]
  start target file
  built target tree_file [0 seconds]
  start target tree
  built target file [0.001 seconds]
  start target data
  built target tree [0.383 seconds]
  built target data [0.242 seconds]
  start branch subsample_62e81964
  start branch subsample_b4bfd040
  built branch subsample_62e81964 [0.076 seconds]
  start branch sim_data_c9704089
  start branch subsample_886a580b
  built branch subsample_b4bfd040 [0.286 seconds]
  built branch sim_data_c9704089 [0.315 seconds]
  start branch ps_data_25e8574f
  built branch subsample_886a580b [0.241 seconds]
  start branch sim_data_e0dd2595
  built branch ps_data_25e8574f [0.241 seconds]
  start branch tbss_410784a2
  built branch sim_data_e0dd2595 [0.243 seconds]
  start branch ps_data_c8fcfa9e
  built branch ps_data_c8fcfa9e [0.252 seconds]
  start branch tbss_cb746b36
  built branch tbss_410784a2 [2.244 seconds]
  start branch subsample_9cda9fa1
  built branch subsample_9cda9fa1 [0.045 seconds]
  start branch sim_data_dc21cee1
  built branch sim_data_dc21cee1 [0.141 seconds]
  start branch ps_data_c2298b02
  built branch tbss_cb746b36 [2.684 seconds]
  start branch subsample_d8835e12
  built branch ps_data_c2298b02 [0.26 seconds]
  start branch tbss_40a1c294
  built branch subsample_d8835e12 [0.056 seconds]
```

```
start branch sim_data_85411e4b
built branch sim_data_85411e4b [0.173 seconds]
start branch ps_data_c1d5291e
built branch ps_data_c1d5291e [0.259 seconds]
start branch tbss_bb2d41e4
built branch tbss_40a1c294 [2.087 seconds]
start branch subsample_2db33782
built branch subsample_2db33782 [0.046 seconds]
start branch sim_data_c5f5adf9
built branch sim_data_c5f5adf9 [0.141 seconds]
start branch ps_data_e66d3b3b
built branch ps_data_e66d3b3b [0.258 seconds]
start branch tbss_81b8b7a8
built branch tbss_bb2d41e4 [2.913 seconds]
start branch subsample_e2199db5
built branch subsample_e2199db5 [0.057 seconds]
start branch sim_data_20ee2357
built branch sim_data_20ee2357 [0.158 seconds]
start branch ps_data_8115835d
built branch ps_data_8115835d [0.266 seconds]
start branch tbss_d06d9655
built branch tbss_81b8b7a8 [2.11 seconds]
start branch subsample_38a79088
built branch subsample_38a79088 [0.043 seconds]
start branch sim_data_85fd2479
built branch sim_data_85fd2479 [0.155 seconds]
start branch ps_data_761d80a0
built branch ps_data_761d80a0 [0.248 seconds]
start branch tbss_306f3d59
built branch tbss_d06d9655 [2.706 seconds]
start branch subsample_fb5d7868
built branch subsample_fb5d7868 [0.045 seconds]
start branch sim_data_7e93bdc4
built branch sim_data_7e93bdc4 [0.149 seconds]
start branch ps_data_dda3b972
built branch ps_data_dda3b972 [0.253 seconds]
start branch tbss_bd620b9b
built branch tbss_306f3d59 [2.717 seconds]
start branch sim_data_0cea77f1
built branch sim_data_0cea77f1 [0.141 seconds]
start branch ps_data_60f56dbc
built branch ps_data_60f56dbc [0.239 seconds]
start branch tbss_10f16beb
```

```
  built branch tbss_bd620b9b [2.384 seconds]
  start branch subsample_1b97e11f
  built branch subsample_1b97e11f [0.046 seconds]
  built pattern subsample
  start branch sim_data_788d8f30
  built branch sim_data_788d8f30 [0.167 seconds]
  built pattern sim_data
  start branch ps_data_30a6f5b3
  built branch ps_data_30a6f5b3 [0.249 seconds]
  built pattern ps_data
  start branch tbss_2b6cf182
  built branch tbss_10f16beb [2.389 seconds]
  built branch tbss_2b6cf182 [1.982 seconds]
  built pattern tbss
  start target final_vector
  built target final_vector [0.001 seconds]
  start target final_power
  built target final_power [0.001 seconds]
  end pipeline [24.169 seconds]
Warning message:
10 targets produced warnings. Run targets::tar_meta(fields = warnings, complete_only = TRUE)

[1] "Final power: 0.2"
```