Running multiple analyses at once using the CaseCrossover package

Martijn J. Schuemie

2017-10-25

Contents

1 Introduction

In this vignette we focus on running several different analyses on several exposure-outcome-nesting cohort triplets This can be useful when we want to explore the sensitivity to analyses choices, include controls, or run an experiment similar to the OMOP experiment to empirically identify the optimal analysis choices for a particular research question.

This vignette assumes you are already familiar with the `CaseCrossover` package and are able to perform single studies. We will walk through all the steps needed to perform an exemplar set of analyses, and we have selected the well-studied topic of the effect of nonsteroidal anti-inflammatory drugs (NSAIDs) on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one NSAID: diclofenac. We will execute various variations of an analysis for the primary exposure pair and a large set of negative control exposures.

2 General approach

The general approach to running a set of analyses is that you specify all the function arguments of the functions you would normally call, and create sets of these function arguments. The final models as well as intermediate data objects will all be saved to disk for later extraction.

An analysis will be executed by calling these functions in sequence:

1. `getDbCaseCrossoverData()`
2. `selectSubjectsToInclude()`
3. `getExposureStatus()`

4. `fitCaseCrossoverModel()`

When you provide several analyses to the `CaseCrossover` package, it will determine whether any of the analyses and exposure-outcome-nesting triplets cohort triplets have anything in common, and will take advantage of this fact. For example, if we specify several exposure-outcome-nesting triplets with the same outcome and nesting cohort, the data for the cases will be extracted only once.

The function arguments you need to define have been divided into four groups:

1. **Hypothesis of interest**: arguments that are specific to a hypothesis of interest, in the case of the case-crossover design this is a combination of exposure, outcome, and optionally a cohort in which the analysis is nested.
2. **Analyses**: arguments that are not directly specific to a hypothesis of interest, such as the washout window, whether to focus on first outcomes or all, or whether to adjust for time trends in exposure.
3. Arguments that are the output of a previous function in the `CaseCrossover` package, such as the `caseCrossoverData` argument of the `selectSubjectsToInclude` function. These cannot be specified by the user.
4. Arguments that are specific to an environment, such as the connection details for connecting to the server, and the name of the schema holding the CDM data.

3 Preparation for the example

We need to tell R how to connect to the server where the data are. `CaseCrossover` uses the `DatabaseConnector` package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")

cdmDatabaseSchema <- "my_cdm_data"
cohortDatabaseSchema <- "my_work_schema"
cohortTable <- "vignette_cohorts"
```

The last two lines define the `cdmDatabaseSchema` and `cohortDatabaseSchema` variables. We'll use these later to tell R where the data in CDM format live, and where we want to store the outcome and nesting cohorts. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

We also need to prepare our exposures, outcomes and nesting cohorts of interest. The `drug_era` table in the OMOP Common Data Model already contains prespecified cohorts of users at the ingredient level, so we will use that for the exposures here. Note that we can also use custom-defined exposure cohorts, such as those created in ATLAS. For the outcomes, we want to restrict our analysis only to those events that are recorded in an inpatient setting, so we will need to create a custom cohort table. For this example, we are only interested in GI bleed (concept ID 192671). Some of the analyses we'd like to nest in a population with previous diagnoses of rheumatoid arthritis, so we will need to create a custom cohort for this as well.

We create a text file called *vignette.sql* with the following content:

```
/***********************************
File vignette.sql
***********************************/

IF OBJECT_ID('@cohortDatabaseSchema.@cohortTable', 'U') IS NOT NULL
```

```
  DROP TABLE @cohortDatabaseSchema.@cohortTable;

SELECT 1 AS cohort_definition_id,
    condition_start_date AS cohort_start_date,
    condition_end_date AS cohort_end_date,
    condition_occurrence.person_id AS subject_id
INTO @cohortDatabaseSchema.@cohortTable
FROM @cdmDatabaseSchema.condition_occurrence
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON condition_occurrence.visit_occurrence_id = visit_occurrence.visit_occurrence_id
WHERE condition_concept_id IN (
        SELECT descendant_concept_id
        FROM @cdmDatabaseSchema.concept_ancestor
        WHERE ancestor_concept_id = 192671 -- GI - Gastrointestinal haemorrhage
        )
    AND visit_occurrence.visit_concept_id IN (9201, 9203);

INSERT INTO @cohortDatabaseSchema.@cohortTable
(cohort_definition_id, cohort_start_date, cohort_end_date, subject_id)
SELECT 2 AS cohort_definition_id,
    MIN(condition_start_date) AS cohort_start_date,
    NULL AS cohort_end_date,
    person_id AS subject_id
FROM @cdmDatabaseSchema.condition_occurrence
WHERE condition_concept_id IN (
        SELECT descendant_concept_id
        FROM @cdmDatabaseSchema.concept_ancestor
        WHERE ancestor_concept_id = 80809 -- rheumatoid arthritis
        )
GROUP BY person_id;
```

This is parameterized SQL which can be used by the `SqlRender` package. We use parameterized SQL so we do not have to pre-specify the names of the CDM and result schemas. That way, if we want to run the SQL on a different schema, we only need to change the parameter values; we do not have to change the SQL code. By also making use of translation functionality in `SqlRender`, we can make sure the SQL code can be run in many different environments.

```
library(SqlRender)
sql <- readSql("vignette.sql")
sql <- renderSql(sql,
                 cdmDatabaseSchema = cdmDatabaseSchema,
                 cohortDatabaseSchema = cohortDatabaseSchema,
                 cohortTable = cohortTable)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

connection <- connect(connectionDetails)
executeSql(connection, sql)
```

In this code, we first read the SQL from the file into memory. In the next line, we replace the two parameter names with the actual values. We then translate the SQL into the dialect appropriate for the DBMS we already specified in the `connectionDetails`. Next, we connect to the server, and submit the rendered and translated SQL.

4 Specifying hypotheses of interest

The first group of arguments define the exposure, outcome, and optionally the nesting cohort. Here we demonstrate how to create a list of exposure-outcome-nesting cohort triplets:

```
negativeControls <- c(705178,
                      705944,
                      710650,
                      714785,
                      719174,
                      719311,
                      735340,
                      742185,
                      780369,
                      781182,
                      924724,
                      990760,
                      1110942,
                      1111706,
                      1136601,
                      1317967,
                      1501309,
                      1505346,
                      1551673,
                      1560278,
                      1584910,
                      19010309,
                      19044727,
                      40163731)
diclofenac <- 1124300
giBleed <- 1
rheumatoidArthritis <- 2

exposureOutcomeNcList <- list()
for (exposureId in c(diclofenac, negativeControls)) {
  exposureOutcomeNc <- createExposureOutcomeNestingCohort(exposureId = exposureId,
                                                          outcomeId = giBleed,
                                                          nestingCohortId = rheumatoidArthritis)
  exposureOutcomeNcList[[length(exposureOutcomeNcList) + 1]] <- exposureOutcomeNc
}
```

We defined the outcome of interest to be the custom cohort with ID 1 we defined in the SQL above. The exposures include diclofenac (concept ID 1124300) and a large number of negative control exposures. For each hypothesis we specify the same nesting cohort with ID 2, as defined in the SQL above.

A convenient way to save `exposureOutcomeNcList` to file is by using the `saveExposureOutcomeNestingCohortList` function, and we can load it again using the `loadExposureOutcomeNestingCohortList` function.

5 Specifying analyses

The second group of arguments are not specific to a hypothesis of interest, and comprise the majority of arguments. For each function that will be called during the execution of the analyses, a companion function is available that has (almost) the same arguments. For example, for the `getDbCaseCrossoverData()` function

there is the `createGetDbCaseCrossoverDataArgs()` function. These companion functions can be used to create the arguments to be used during execution:

```
getDbCaseCrossoverDataArgs1 <- createGetDbCaseCrossoverDataArgs(useNestingCohort = FALSE)

selectSubjectsToIncludeArgs1 <- createSelectSubjectsToIncludeArgs(firstOutcomeOnly = FALSE,
                                                                  washoutPeriod = 180)

getExposureStatusArgs1 <- createGetExposureStatusArgs(firstExposureOnly = FALSE,
                                                      riskWindowStart = 0,
                                                      riskWindowEnd = 0,
                                                      controlWindowOffsets = -30)
```

Any argument that is not explicitly specified by the user will assume the default value specified in the function. Note that in this example, even though we specified a nesting cohort for each exposure-outcome-nesting cohort triplet, we have specified `useNestingCohort = FALSE`, meaning we will not nest this analysis but rather use the entire population to draw cases and controls.

We can now combine the arguments for the various functions into a single analysis:

```
ccrAnalysis1 <- createCcrAnalysis(analysisId = 1,
                                  description = "Simple case-crossover",
                                  getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs1,
                                  selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs1,
                                  getExposureStatusArgs = getExposureStatusArgs1)
```

Note that we have assigned an analysis ID (1) to this set of arguments. We can use this later to link the results back to this specific set of choices. We also include a short description of the analysis.

We can easily create more analyses, for example by using nesting and correcting for time trends in exposure (the time-case-control design):

```
getDbCaseCrossoverDataArgs2 <- createGetDbCaseCrossoverDataArgs(useNestingCohort = TRUE,
                                                                getTimeControlData = TRUE,
                                                                getVisits = TRUE)

ccrAnalysis2 <- createCcrAnalysis(analysisId = 2,
                                  description = "Nested case-crossover",
                                  getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs2,
                                  selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs1,
                                  getExposureStatusArgs = getExposureStatusArgs1)

matchingCriteria1 <- createMatchingCriteria(matchOnAge = TRUE,
                                            ageCaliper = 2,
                                            matchOnGender = TRUE)

selectSubjectsToIncludeArgs2 <- createSelectSubjectsToIncludeArgs(firstOutcomeOnly = FALSE,
                                                                  washoutPeriod = 180,
                                                                  matchingCriteria = matchingCriteria1)

ccrAnalysis3 <- createCcrAnalysis(analysisId = 3,
                                  description = "Nested case-time-control, matching on age and gender",
                                  getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs2,
                                  selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs2,
                                  getExposureStatusArgs = getExposureStatusArgs1)

matchingCriteria2 <- createMatchingCriteria(matchOnAge = TRUE,
```

```
                                            ageCaliper = 2,
                                            matchOnGender = TRUE,
                                            matchOnVisitDate = TRUE)

selectSubjectsToIncludeArgs3 <- createSelectSubjectsToIncludeArgs(firstOutcomeOnly = FALSE,
                                                                  washoutPeriod = 180,
                                                                  matchingCriteria = matchingCriteria2)

ccrAnalysis4 <- createCcrAnalysis(analysisId = 4,
                                  description = "Nested case-time-control, matching on age, gender, and vi
                                  getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs2,
                                  selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs3,
                                  getExposureStatusArgs = getExposureStatusArgs1)
```

These analyses can be combined in a list:

```
ccrAnalysisList <- list(ccrAnalysis1, ccrAnalysis2, ccrAnalysis3, ccrAnalysis4)
```

A convenient way to save `ccrAnalysisList` to file is by using the `saveCcrAnalysisList` function, and we can load it again using the `loadCcrAnalysisList` function.

5.1 Exposure, outcome, and nesting cohort selection strategies

Often we would like to evaluate different definitions of the exposure and/or outcome, or we want to consider different strategies for identifying the nesting cohort. We could include these by created extra exposure-outcome-nesting cohort triplets, but that would mean that all defined analyses would be executed against these variations of the definitions, and this may not be what we want. Perhaps we would like to define just a single sensitivity analyses with a different outcome definition, in which case we could argue that the strategy of selecting the outcome becomes part of the analysis.

In such a case, we can define the multiple strategies using a list:

```
outcomeIds = list(narrowDefinition = 1,
                  broadDefinition = 11)

exposureOutcomeNc <- createExposureOutcomeNestingCohort(exposureId = 1124300,
                                                        outcomeId = outcomeIds,
                                                        nestingCohortId = 2)
```

When we specify an analysis, we can then refer to one definition or the other:

```
ccrAnalysis1A <- createCcrAnalysis(analysisId = 1,
                                   description = "Simple case-crossover, using narrow def.",
                                   outcomeType = "narrowDefinition",
                                   getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs1,
                                   selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs1,
                                   getExposureStatusArgs = getExposureStatusArgs1)

ccrAnalysis1B <- createCcrAnalysis(analysisId = 2,
                                   description = "Simple case-crossover, using broad def.",
                                   outcomeType = "broadDefinition",
                                   getDbCaseCrossoverDataArgs = getDbCaseCrossoverDataArgs1,
                                   selectSubjectsToIncludeArgs = selectSubjectsToIncludeArgs1,
                                   getExposureStatusArgs = getExposureStatusArgs1)
```

```
ccrAnalysisList2 <- list(ccrAnalysis1A, ccrAnalysis1B)
```

In this example, the first analysis (analysisId = 1) will use cohort definition 1 as outcome, whilst the second analysis analysis (analysisId = 2) will use cohort definition 11 as outcome.

The same mechanism can be used to specifiy types for the exposureId and nestingCohortId.

6 Executing multiple analyses

We can now run the analyses against the hypotheses of interest using the `runCcrAnalyses()`function. This function will run all specified analyses against all hypotheses of interest, meaning that the total number of outcome models is `length(ccrAnalysisList) * length(exposureOutcomeNcList)`.

```
result <- runCcrAnalyses(connectionDetails = connectionDetails,
                         cdmDatabaseSchema = cdmDatabaseSchema,
                         oracleTempSchema = cdmDatabaseSchema,
                         exposureDatabaseSchema = cdmDatabaseSchema,
                         exposureTable = "drug_era",
                         outcomeDatabaseSchema = cohortDatabaseSchema,
                         outcomeTable = cohortTable,
                         nestingCohortDatabaseSchema = cohortDatabaseSchema,
                         nestingCohortTable = cohortTable,
                         outputFolder = outputFolder,
                         exposureOutcomeNestingCohortList = exposureOutcomeNcList,
                         ccrAnalysisList = ccrAnalysisList,
                         getDbCaseCrossoverDataThreads = 1,
                         selectSubjectsToIncludeThreads = 4,
                         getExposureStatusThreads = 3,
                         fitCaseCrossoverModelThreads = 4)
```

In the code above, we provide the arguments for connecting to the database, which schemas and tables to use, as well as the analyses and hypotheses of interest. The `outputFolder` specifies where the outcome models and intermediate files will be written. We also instruct `CaseCrossover` to use multiple threads for various stages in the analyses, meaning these will be executed in parallel on multiple CPUs in the computer. Multithreading can significantly reduce execution time, but will require more system resources such as memory and temporary disk space.

6.1 Restarting

If for some reason the execution was interrupted, you can restart by re-issuing the `runCcrAnalyses()` command. Any intermediate and final products that have already been completed and written to disk will be skipped.

7 Retrieving the results

The result of the `runCcrAnalyses()` is a data frame with one row per exposure-outcome-nesting cohort-analysis combination. It provides the file names of the intermediate and end-result files that were constructed. For example, we can retrieve the fitted model for the combination of our drug of interest, outcome, and first analysis:

```
ccrModelFile <- result$modelFile[result$exposureId == 1124300 &
                                 result$outcomeId == 1 &
                                 result$analysisId == 1]
ccModel <- readRDS(ccrModelFile)
summary(ccrModel)
```

```
#> Case-Control fitted model
#> Status: OK
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment 1.076923  0.976117  1.188140 0.074108  0.0501
#>
#> Counts
#>        Cases Controls Exposed cases Exposed controls     NA     NA     NA
#> Count 314904        0        314904                0   1849   1790      0
#>       NA
#> Count  0
```

Note that some of the file names will appear several times in the table. For example, analyses 2-4 share the same ccrData object.

We can create a summary of the results using `summarizeCcrAnalyses()`:

```
analysisSum <- summarizeCcrAnalyses(result)
head(analysisSum)
```

```
#>   analysisId exposureId nestingCohortId outcomeId        rr    ci95lb
#> 1          1    1124300               2         1 1.0769231 0.9761171
#> 2          1     705178               2         1 0.9523810 0.6176690
#> 3          1     705944               2         1 1.0347222 0.8229128
#> 4          1     710650               2         1 0.9130435 0.5053346
#> 5          1     714785               2         1 1.1666667 0.9528137
#> 6          1     719174               2         1 1.0000000 0.8646053
#>     ci95ub         p  cases controls casesControlWindows
#> 1 1.188140 0.1394348 314904        0              314904
#> 2 1.468472 0.8252161 314904        0              314904
#> 3 1.301049 0.7702186 314904        0              314904
#> 4 1.649696 0.7631039 314904        0              314904
#> 5 1.428518 0.1356734 314904        0              314904
#> 6 1.156597 1.0000000 314904        0              314904
#>   controlsControlWindows exposedCasesCaseWindow exposedCasesControlWindow
#> 1                      0                  1849                      1790
#> 2                      0                   285                       287
#> 3                      0                   798                       793
#> 4                      0                   102                       104
#> 5                      0                  1412                      1383
#> 6                      0                  2122                      2122
#>   exposedControlsCaseWindow exposedControlsControlWindow         logRr
#> 1                         0                            0  7.410797e-02
#> 2                         0                            0 -4.879016e-02
#> 3                         0                            0  3.413301e-02
#> 4                         0                            0 -9.097178e-02
#> 5                         0                            0  1.541507e-01
#> 6                         0                            0 -2.962587e-16
#>      seLogRr
```

```
#> 1 0.05014414
#> 2 0.22092878
#> 3 0.11685826
#> 4 0.30182331
#> 5 0.10331135
#> 6 0.07422696
```

This tells us, per exposure-outcome-nesting cohort-analysis combination, the estimated relative risk and 95% confidence interval, as well as the number of cases, controls (for time-case-control), and number of those that were exposed to the drug in the various windows.

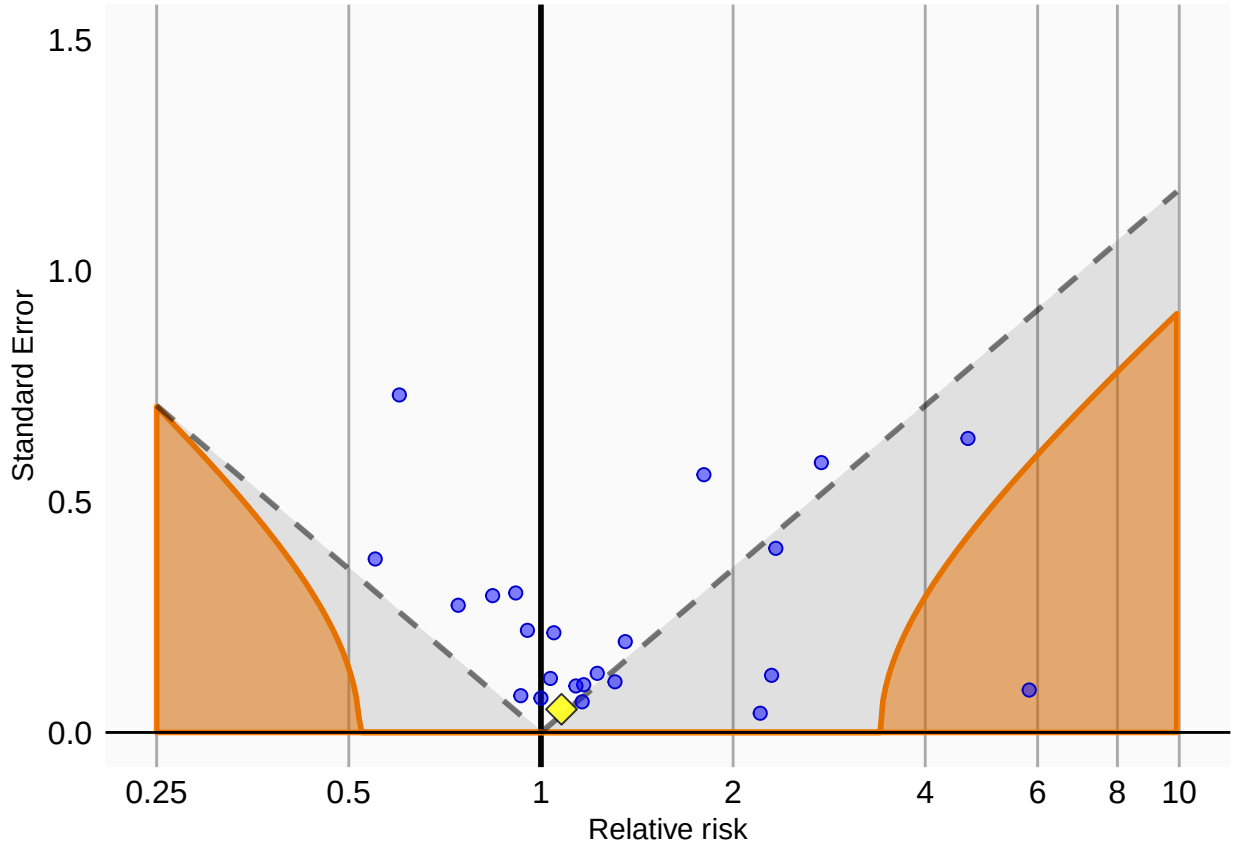
7.1 Empirical calibration

Now that we have produced estimates for all outcomes including our negative controls, we can perform empirical calibration to estimate the bias of the various analyses included in our study. We will create the calibration effect plots for every analysis ID. In each plot, the blue dots represent our negative control exposures, and the yellow diamond represents our exposure of interest: diclofenac. An unbiased, well-calibrated analysis should have 95% of the negative controls between the dashed lines (ie. 95% should have p $>$ .05).

```
library(EmpiricalCalibration)

# Analysis 1: Simple case-crossover
negCons <- analysisSum[analysisSum$analysisId == 1 & analysisSum$exposureId != 1124300, ]
ei <-  analysisSum[analysisSum$analysisId == 1 & analysisSum$exposureId == 1124300, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(logRrNegatives = negCons$logRr,
                      seLogRrNegatives = negCons$seLogRr,
                      logRrPositives = ei$logRr,
                      seLogRrPositives = ei$seLogRr,
                      null)
```

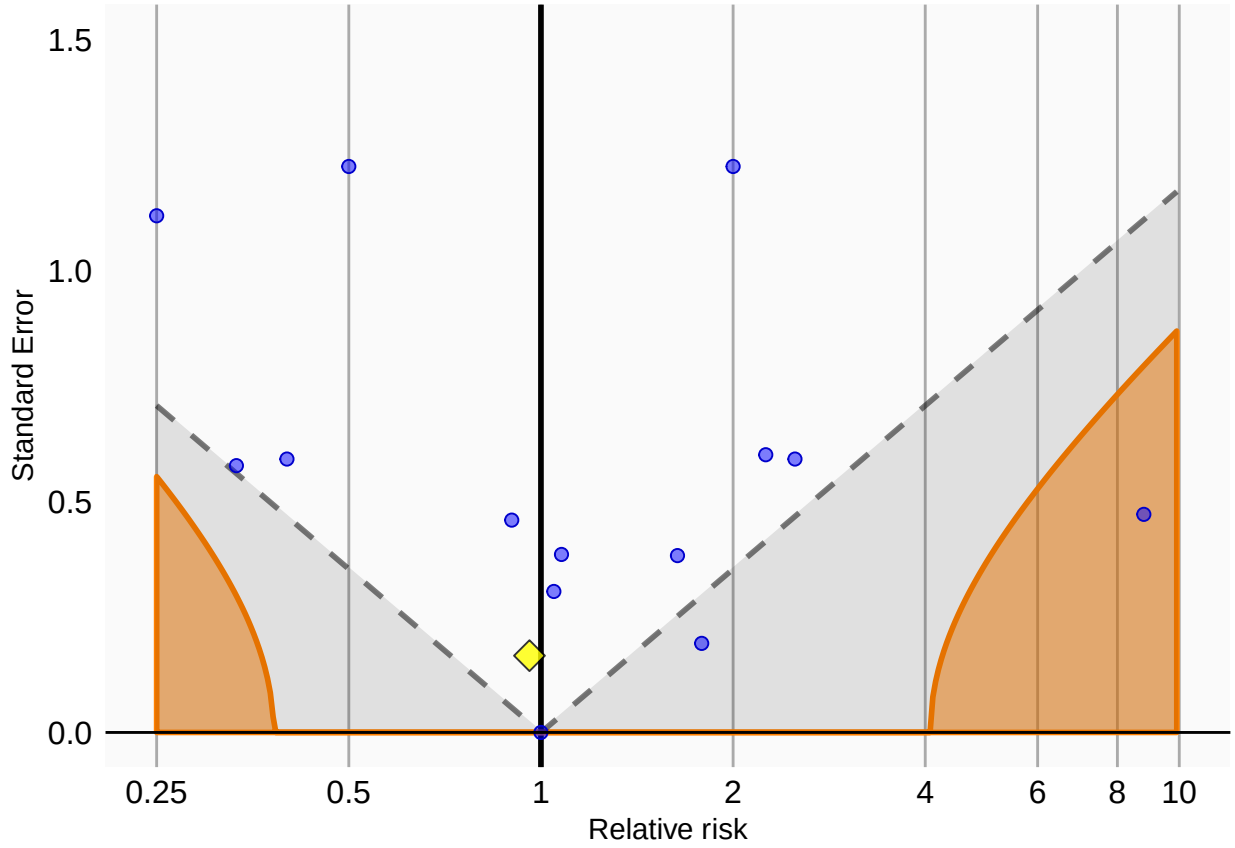
```
#> Warning: Removed 1 rows containing missing values (geom_point).
```




```
# Analysis 2: Nesting in rheumatoid arthritis
negCons <- analysisSum[analysisSum$analysisId == 2 & analysisSum$exposureId != 1124300, ]
ei <-  analysisSum[analysisSum$analysisId == 2 & analysisSum$exposureId == 1124300, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(logRrNegatives = negCons$logRr,
                      seLogRrNegatives = negCons$seLogRr,
                      logRrPositives = ei$logRr,
                      seLogRrPositives = ei$seLogRr,
                      null)
```

```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with NA
#> standard error detected. Removing before fitting null distribution
```

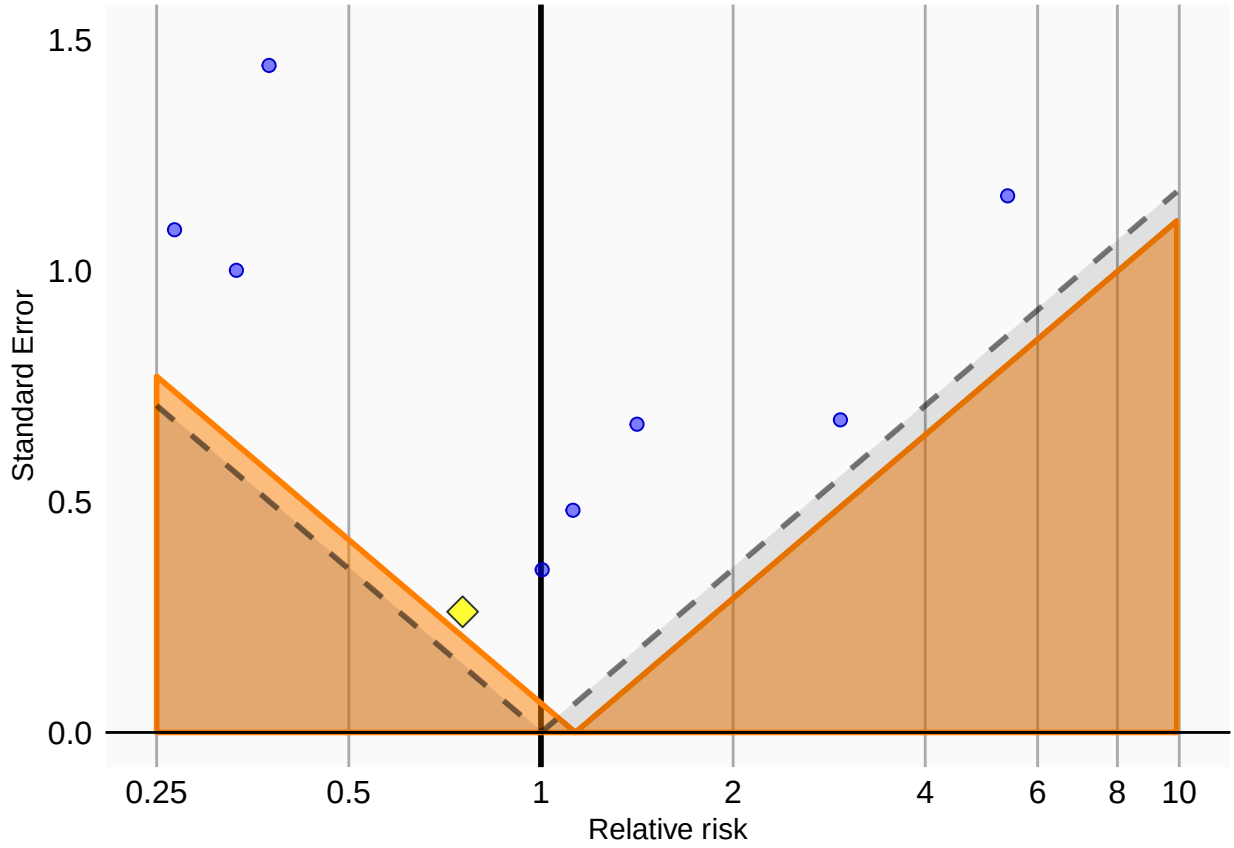
```
#> Warning: Removed 10 rows containing missing values (geom_point).
```




```
# Analysis 3: Nested case-time-control, matching on age and gender
negCons <- analysisSum[analysisSum$analysisId == 3 & analysisSum$exposureId != 1124300, ]
ei <-  analysisSum[analysisSum$analysisId == 3 & analysisSum$exposureId == 1124300, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(logRrNegatives = negCons$logRr,
                      seLogRrNegatives = negCons$seLogRr,
                      logRrPositives = ei$logRr,
                      seLogRrPositives = ei$seLogRr,
                      null)
```

```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with NA
#> standard error detected. Removing before fitting null distribution
```

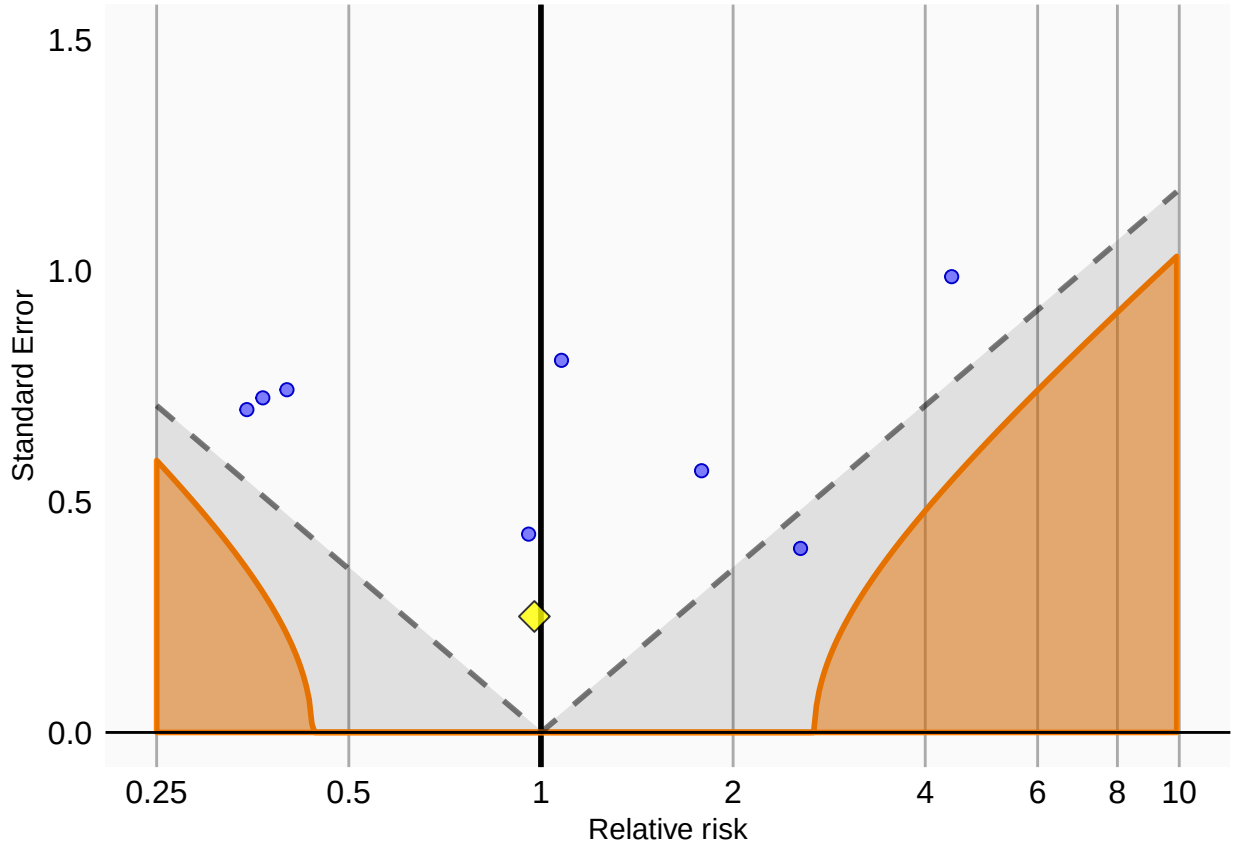
```
#> Warning: Removed 16 rows containing missing values (geom_point).
```




```
# Analysis 4: Nested case-time-control, matching on age, gender, and visit
negCons <- analysisSum[analysisSum$analysisId == 4 & analysisSum$exposureId != 1124300, ]
ei <-  analysisSum[analysisSum$analysisId == 4 & analysisSum$exposureId == 1124300, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(logRrNegatives = negCons$logRr,
                      seLogRrNegatives = negCons$seLogRr,
                      logRrPositives = ei$logRr,
                      seLogRrPositives = ei$seLogRr,
                      null)
```

```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with NA
#> standard error detected. Removing before fitting null distribution
```

```
#> Warning: Removed 16 rows containing missing values (geom_point).
```



Acknowledgments

Considerable work has been dedicated to provide the `CaseCrossover` package.

```
citation("CaseCrossover")
```

```
#>
#> To cite package 'CaseCrossover' in publications use:
#>
#>   Martijn Schuemie (2017). CaseCrossover: Case-Crossover. R
#>   package version 1.0.1.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {CaseCrossover: Case-Crossover},
#>     author = {Martijn Schuemie},
#>     year = {2017},
#>     note = {R package version 1.0.1},
#>   }
#>
#> ATTENTION: This citation information has been auto-generated from
#> the package DESCRIPTION file and may need manual editing, see
#> 'help("citation")'.
```

Further, `CaseCrossover` makes extensive use of the `Cyclops` package.

```
citation("Cyclops")
```

```
#>
#> To cite Cyclops in publications use:
#>
#> Suchard MA, Simpson SE, Zorych I, Ryan P and Madigan D (2013).
#> "Massive parallelization of serial inference algorithms for
#> complex generalized linear models." _ACM Transactions on Modeling
#> and Computer Simulation_, *23*, pp. 10. <URL:
#> http://dl.acm.org/citation.cfm?id=2414791>.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Article{,
#>     author = {M. A. Suchard and S. E. Simpson and I. Zorych and P. Ryan and D. Madigan},
#>     title = {Massive parallelization of serial inference algorithms for complex generalized linear m
#>     journal = {ACM Transactions on Modeling and Computer Simulation},
#>     volume = {23},
#>     pages = {10},
#>     year = {2013},
#>     url = {http://dl.acm.org/citation.cfm?id=2414791},
#>   }
```

This work is supported in part through the National Science Foundation grant IIS 1251151.