Running multiple analyses at once using the CohortMethod package

Martijn J. Schuemie, Marc A. Suchard and Patrick Ryan

2016-02-29

Contents

1 Introduction

In this vignette we focus on running several different analyses on several drug-comparator-outcome combinations. 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 `CohortMethod` 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 coxibs versus non-selective nonsteroidal anti-inflammatory drugs (NSAIDs) on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one coxib – celecoxib – and one non-selective NSAID – diclofenac. We will execute various variations of an analysis for the primary outcome and a large set of negative control outcomes.

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 outcome 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. `getDbCohortMethodData()`
2. `createPs()` (optional)
3. `trimByPs()` or `trimByPsToEquipoise()` (optional)
4. `matchOnPs()`, `matchOnPsAndCovariates()`, `stratifyByPs()`, or `stratifyByPsAndCovariates()` (optional)
5. `computeCovariateBalance()` (optional)
6. `fitOutcomeModel()` (optional)

When you provide several analyses to the `CohortMethod` package, it will determine whether any of the analyses have anything in common, and will take advantage of this fact. For example, if we specify several analyses that only differ in the way the outcome model is fitted, then `CohortMethod` will extract the data and fit the propensity model only once, and re-use this in all the analyses.

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 cohort method this is a combination of drug, comparator, and outcome (and indication if required).
2. **Analyses**: arguments that are not directly specific to a hypothesis of interest, such as the washout window, whether to include drugs as covariates, etc.
3. Arguments that are the output of a previous function in the `CohortMethod` package, such as the `cohortMethodData` argument of the `createPs` 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.

There are a three arguments (`exclusionConceptIds`, `excludedCovariateConceptIds`, and `includedCovariateConceptIds` of the `getDbCohortMethodData()` function) that can be argued to be part both of group 1 and 2. For example, our design in general could exclude cancer patients, and we would specify this as part of the analysis using the `exclusionConceptIds` argument. In other situations we might want to exclude people with prior exposure to the target drug or any related drug, and we would also use the `exclusionConceptIds`. These arguments are therefore present in both groups, and when executing the analysis the union of the two lists of concept IDs will be used.

3 Preparation for the example

We need to tell R how to connect to the server where the data are. `CohortMethod` 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"
resultsDatabaseSchema <- "my_results"
cdmVersion <- "5"
```

The last three lines define the `cdmDatabaseSchema` and `resultSchema` variables,as well as the CDM version. We'll use these later to tell R where the data in CDM format live, where we want to write intermediate and result tables, and what version CDM is used. 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 and outcomes 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. For the outcomes, we want to restrict our analysis only to those outcomes that are recorded in an inpatient setting, so we will need to create a custom cohort table. For this example, we want to include GI bleed (concept ID 192671) as well as a set of 35 negative controls. Negative controls are defined as those outcomes where there is no evidence that either the target drug (celexocib) or comparator drug (diclofenac) causes the outcome.

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

```sql
/**********************************
File VignetteOutcomes.sql
***********************************/
IF OBJECT_ID('@resultsDatabaseSchema.outcomes', 'U') IS NOT NULL
  DROP TABLE @resultsDatabaseSchema.outcomes;

SELECT ancestor_concept_id 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 @resultsDatabaseSchema.outcomes
FROM @cdmDatabaseSchema.condition_occurrence
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON condition_occurrence.visit_occurrence_id = visit_occurrence.visit_occurrence_id
INNER JOIN @cdmDatabaseSchema.concept_ancestor
    ON condition_concept_id = descendant_concept_id
WHERE ancestor_concept_id IN (192671, 24609, 29735, 73754, 80004, 134718, 139099,
141932, 192367, 193739, 194997, 197236, 199074, 255573, 257007, 313459, 314658,
316084, 319843, 321596, 374366, 375292, 380094, 433753, 433811, 436665, 436676,
436940, 437784, 438134, 440358, 440374, 443617, 443800, 4084966, 4288310)
    AND visit_occurrence.visit_concept_id IN (9201, 9203);
```

Note for CDM V4 'visit_concept_id' should be 'place_of_service_concept_id', and 'cohort_definition_id' should be 'cohort_concept_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.

```r
library(SqlRender)
sql <- readSql("VignetteOutcomes.sql")
sql <- renderSql(sql,
                 cdmDatabaseSchema = cdmDatabaseSchema,
```

```
                 resultsDatabaseSchema = resultsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$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 drug, comparator, outcome, and optionally the indication in which to
nest the study. Here we demonstrate how to create one set, and add that set to a list:

```
# Get all NSAID Concept IDs for exclusion:
sql <- paste("SELECT concept_id",
             "FROM @cdmDatabaseSchema.concept_ancestor",
             "INNER JOIN @cdmDatabaseSchema.concept",
             "ON descendant_concept_id = concept_id",
             "WHERE ancestor_concept_id = 21603933")
sql <- SqlRender::renderSql(sql, cdmDatabaseSchema = cdmDatabaseSchema)$sql
sql <- SqlRender::translateSql(sql, targetDialect = connectionDetails$dbms)$sql
nsaids <- querySql(connection, sql)
nsaids <- nsaids$CONCEPT_ID

dcos <- createDrugComparatorOutcomes(targetId = 1118084,
                                     comparatorId = 1124300,
                                     exclusionConceptIds = nsaids,
                                     excludedCovariateConceptIds = nsaids,
                                     outcomeIds = c(192671, 29735, 140673, 197494,
                                                    198185, 198199, 200528, 257315,
                                                    314658, 317376, 321319, 380731,
                                                    432661, 432867, 433516, 433701,
                                                    433753, 435140, 435459, 435524,
                                                    435783, 436665, 436676, 442619,
                                                    444252, 444429, 4131756, 4134120,
                                                    4134454, 4152280, 4165112, 4174262,
                                                    4182210, 4270490, 4286201, 4289933))

drugComparatorOutcomesList <- list(dcos)
```

We defined the target to be celecoxib (concept ID 1118084), the comparator to be diclofenac (concept ID
1124300), and the outcomes of interest are GI-bleed (concept ID 192671) and a large number of negative
control outcomes. In this example, we also specify that people with prior exposure to any NSAID should be
excluded when investigating these drugs and outcomes. We furthermore specify that NSAIDS cannot be in
the list of covariates. We use some SQL against the vocabulary to find the concept IDs of all NSAIDS.

A convenient way to save `drugComparatorOutcomeList` to file is by using the `saveDrugComparatorOutcomesList`
function, and we can load it again using the `loadDrugComparatorOutcomesList` 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 `trimByPs()` function there is the `createTrimByPsArgs()` function. These companion functions can be used to create the arguments to be used during execution:

```
covarSettings <- createCovariateSettings(useCovariateDemographics = TRUE,
                                          useCovariateConditionOccurrence = TRUE,
                                          useCovariateConditionOccurrence365d = TRUE,
                                          useCovariateConditionOccurrence30d = TRUE,
                                          useCovariateConditionOccurrenceInpt180d = TRUE,
                                          useCovariateConditionEra = TRUE,
                                          useCovariateConditionEraEver = TRUE,
                                          useCovariateConditionEraOverlap = TRUE,
                                          useCovariateConditionGroup = TRUE,
                                          useCovariateDrugExposure = TRUE,
                                          useCovariateDrugExposure365d = TRUE,
                                          useCovariateDrugExposure30d = TRUE,
                                          useCovariateDrugEra = TRUE,
                                          useCovariateDrugEra365d = TRUE,
                                          useCovariateDrugEra30d = TRUE,
                                          useCovariateDrugEraEver = TRUE,
                                          useCovariateDrugEraOverlap = TRUE,
                                          useCovariateDrugGroup = TRUE,
                                          useCovariateProcedureOccurrence = TRUE,
                                          useCovariateProcedureOccurrence365d = TRUE,
                                          useCovariateProcedureOccurrence30d = TRUE,
                                          useCovariateProcedureGroup = TRUE,
                                          useCovariateObservation = TRUE,
                                          useCovariateObservation365d = TRUE,
                                          useCovariateObservation30d = TRUE,
                                          useCovariateObservationCount365d = TRUE,
                                          useCovariateMeasurement365d = TRUE,
                                          useCovariateMeasurement30d = TRUE,
                                          useCovariateMeasurementCount365d = TRUE,
                                          useCovariateMeasurementBelow = TRUE,
                                          useCovariateMeasurementAbove = TRUE,
                                          useCovariateConceptCounts = TRUE,
                                          useCovariateRiskScores = TRUE,
                                          useCovariateRiskScoresCharlson = TRUE,
                                          useCovariateRiskScoresDCSI = TRUE,
                                          useCovariateRiskScoresCHADS2 = TRUE,
                                          useCovariateInteractionYear = FALSE,
                                          useCovariateInteractionMonth = FALSE,
                                          excludedCovariateConceptIds = c(),
                                          deleteCovariatesSmallCount = 100)

getDbCmDataArgs <- createGetDbCohortMethodDataArgs(washoutWindow = 183,
                                                   indicationLookbackWindow = 183,
                                                   studyStartDate = "",
                                                   studyEndDate = "",
                                                   excludeDrugsFromCovariates = FALSE,
```

```
                                                  covariateSettings = covarSettings)

fitOutcomeModelArgs1 <- createFitOutcomeModelArgs(riskWindowStart = 0,
                                                  riskWindowEnd = 30,
                                                  addExposureDaysToEnd = TRUE,
                                                  useCovariates = FALSE,
                                                  modelType = "cox",
                                                  stratifiedCox = FALSE)
```

Any argument that is not explicitly specified by the user will assume the default value specified in the function. We can now combine the arguments for the various functions into a single analysis:

```
cmAnalysis1 <- createCmAnalysis(analysisId = 1,
                                description = "No matching, simple outcome model",
                                getDbCohortMethodDataArgs = getDbCmDataArgs,
                                fitOutcomeModel = TRUE,
                                fitOutcomeModelArgs = fitOutcomeModelArgs1)
```

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 matching or stratification, or by using more sophisticated outcome models:

```
createPsArgs <- createCreatePsArgs() # Using only defaults

matchOnPsArgs <- createMatchOnPsArgs(maxRatio = 100)

cmAnalysis2 <- createCmAnalysis(analysisId = 2,
                                description = "Matching plus simple outcome model",
                                getDbCohortMethodDataArgs = getDbCmDataArgs,
                                createPs = TRUE,
                                createPsArgs = createPsArgs,
                                matchOnPs = TRUE,
                                matchOnPsArgs = matchOnPsArgs,
                                fitOutcomeModel = TRUE,
                                fitOutcomeModelArgs = fitOutcomeModelArgs1)

stratifyByPsArgs <- createStratifyByPsArgs(numberOfStrata = 5)

cmAnalysis3 <- createCmAnalysis(analysisId = 3,
                                description = "Stratification plus simple outcome model",
                                getDbCohortMethodDataArgs = getDbCmDataArgs,
                                createPs = TRUE,
                                createPsArgs = createPsArgs,
                                stratifyByPs = TRUE,
                                stratifyByPsArgs = stratifyByPsArgs,
                                fitOutcomeModel = TRUE,
                                fitOutcomeModelArgs = fitOutcomeModelArgs1)

fitOutcomeModelArgs2 <- createFitOutcomeModelArgs(riskWindowStart = 0,
                                                  riskWindowEnd = 30,
                                                  addExposureDaysToEnd = TRUE,
```

```
                                          useCovariates = FALSE,
                                          modelType = "cox",
                                          stratifiedCox = TRUE)

cmAnalysis4 <- createCmAnalysis(analysisId = 4,
                                description = "Matching plus stratified outcome model",
                                getDbCohortMethodDataArgs = getDbCmDataArgs,
                                createPs = TRUE,
                                createPsArgs = createPsArgs,
                                matchOnPs = TRUE,
                                matchOnPsArgs = matchOnPsArgs,
                                computeCovariateBalance = TRUE,
                                fitOutcomeModel = TRUE,
                                fitOutcomeModelArgs = fitOutcomeModelArgs2)

fitOutcomeModelArgs3 <- createFitOutcomeModelArgs(riskWindowStart = 0,
                                                  riskWindowEnd = 30,
                                                  addExposureDaysToEnd = TRUE,
                                                  useCovariates = TRUE,
                                                  modelType = "cox",
                                                  stratifiedCox = TRUE)

cmAnalysis5 <- createCmAnalysis(analysisId = 5,
                                description = "Matching plus full outcome model",
                                getDbCohortMethodDataArgs = getDbCmDataArgs,
                                createPs = TRUE,
                                createPsArgs = createPsArgs,
                                matchOnPs = TRUE,
                                matchOnPsArgs = matchOnPsArgs,
                                computeCovariateBalance = TRUE,
                                fitOutcomeModel = TRUE,
                                fitOutcomeModelArgs = fitOutcomeModelArgs3)
```

These analyses can be combined in a list:

```
cmAnalysisList <- list(cmAnalysis1, cmAnalysis2, cmAnalysis3, cmAnalysis4, cmAnalysis5)
```

A convenient way to save `cmAnalysisList` to file is by using the `saveCmAnalysisList` function, and we can load it again using the `loadCmAnalysisList` function.

5.1 Target, comparator and indication selection strategies

Often a new-user cohort design is used for comparative effectiveness studies, where the selection of the comparator is part of the hypothesis of interest: 'Does use of drug A lead to an increased risk compared to use of drug B?', where B is the comparator. But sometimes, the design is used for safety assessment: 'Does use of drug A lead to an increased risk?' In this case the comparator is a proxy for the counterfactual of no treatment. For example, we could pick the comparator to be a drug known not to cause the outcome. we can argue that the selection of the comparator then becomes part of the analyses specification, not the hypothesis of interest, and we can have different strategies for selecting a comparator: Do we for instance pick a drug in the same class, or a drug with the same indication?

In the situation where the comparator choice becomes part of the analyses, we can specify multiple comparators per hypothesis of interest by using a list:

```
comparatorIds = list(drugInSameClass = 1124300,
                     drugWithSameIndication = 1125315)

dcos <- createDrugComparatorOutcomes(targetId = 1118084,
                                     comparatorId = comparatorIds,
                                     outcomeIds = 192671)

drugComparatorOutcomesList2 <- list(dcos)
```

When we specify an analysis, we can then refer to one comparator or another:

```
cmAnalysis1 <- createCmAnalysis(analysisId = 1,
                               description = "Analysis using drug in same class",
                               comparatorType = "drugInSameClass",
                               getDbCohortMethodDataArgs = getDbCmDataArgs,
                               createPs = TRUE,
                               createPsArgs = createPsArgs,
                               matchOnPs = TRUE,
                               matchOnPsArgs = matchOnPsArgs,
                               fitOutcomeModel = TRUE,
                               fitOutcomeModelArgs = fitOutcomeModelArgs1)

cmAnalysis2 <- createCmAnalysis(analysisId = 2,
                               description = "Analysis using drug with same indication",
                               comparatorType = "drugWithSameIndication",
                               getDbCohortMethodDataArgs = getDbCmDataArgs,
                               createPs = TRUE,
                               createPsArgs = createPsArgs,
                               matchOnPs = TRUE,
                               matchOnPsArgs = matchOnPsArgs,
                               fitOutcomeModel = TRUE,
                               fitOutcomeModelArgs = fitOutcomeModelArgs1)

cmAnalysisList2 <- list(cmAnalysis1, cmAnalysis2)
```

In this example, the first analysis (analysisID = 1) will use concept 1124300 as comparator, whilst the second analysis analysis (analysisID = 2) will use concept 1125315 as comparator.

The same mechanism can be used to specifiy types for the targetConceptId and the indicationConceptIds.

6 Executing multiple analyses

We can now run the analyses against the hypotheses of interest using the `runCmAnalyses()` function. This function will run all specified analyses against all hypotheses of interest, meaning that the total number of outcome models is `length(cmAnalysisList) * length(drugComparatorOutcomesList)` (if all analyses specify an outcome model should be fitted).

```
result <- runCmAnalyses(connectionDetails = connectionDetails,
                       cdmDatabaseSchema = cdmDatabaseSchema,
                       exposureDatabaseSchema = cdmDatabaseSchema,
                       exposureTable = "drug_era",
```

```
                    outcomeDatabaseSchema = resultsDatabaseSchema,
                    outcomeTable = "outcomes",
                    cdmVersion = cdmVersion,
                    outputFolder = "./CohortMethodOutput",
                    cmAnalysisList = cmAnalysisList,
                    drugComparatorOutcomesList = drugComparatorOutcomesList,
                    getDbCohortMethodDataThreads = 1,
                    createPsThreads = 1,
                    psCvThreads = 10,
                    computeCovarBalThreads = 2,
                    trimMatchStratifyThreads = 10,
                    fitOutcomeModelThreads = 4,
                    outcomeCvThreads = 10)
```

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 `CohortMethod` 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 `runCmAnalyses()` 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 `runCmAnalyses()` is a data frame with one row per drug-target(-indication)-outcome-analysis combination. It provides the file names of the intermediate and end-result files that were constructed. For example, we can retrieve and plot the propensity scores for the combination of our drug, comparator, outcome of intererst, and last analysis:

```
psFile <- result$psFile[result$targetId == 1118084 &
                        result$comparatorId == 1124300 &
                        result$outcomeId == 192671 &
                        result$analysisId == 5]
ps <- readRDS(psFile)
plotPs(ps)
```
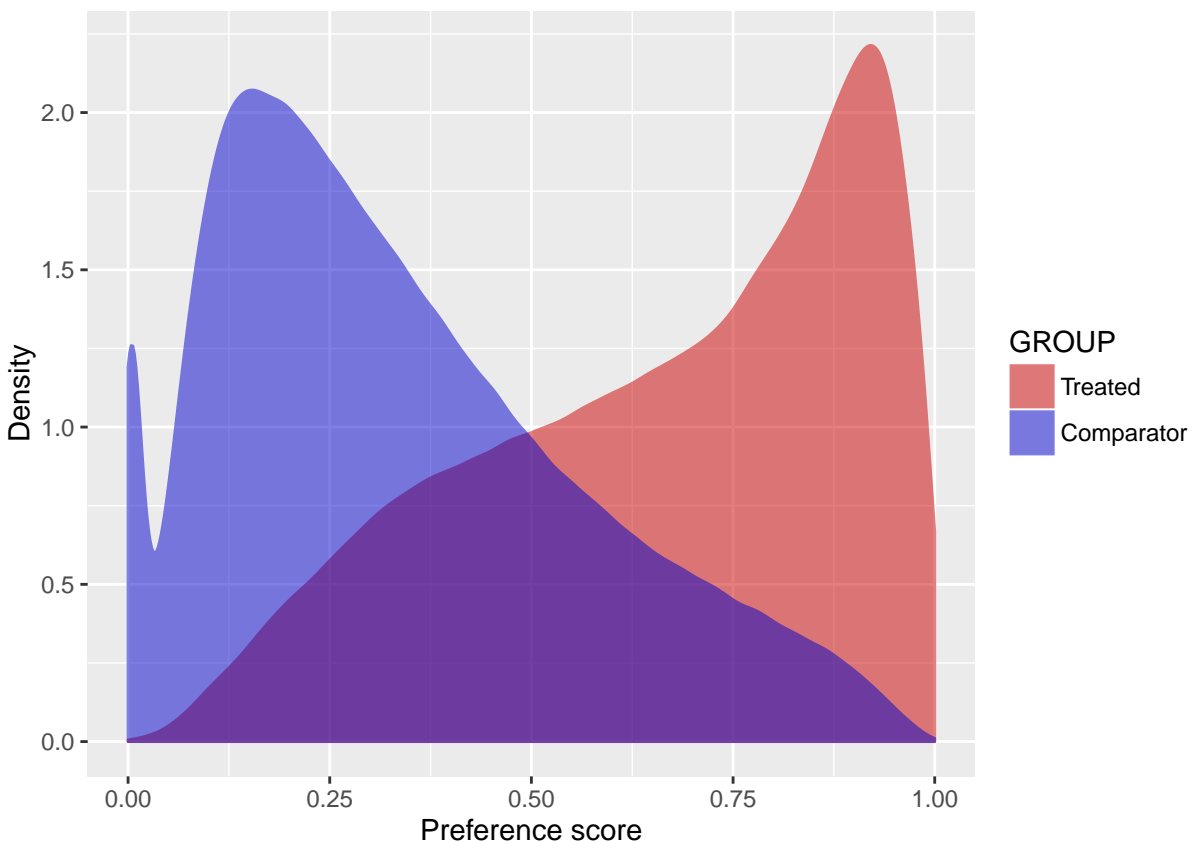
Note that some of the file names will appear several times in the table. For example, analysis 4 and 5 only differ in terms of the outcome model, and will share the same propensity score files.

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

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

```
#>   analysisId targetId comparatorId indicationConceptIds outcomeId
#> 1          1  1118084      1124300                            192671
#> 2          1  1118084      1124300                             24609
#> 3          1  1118084      1124300                             29735
#> 4          1  1118084      1124300                             73754
#> 5          1  1118084      1124300                             80004
#> 6          1  1118084      1124300                            134718
#>          rr      ci95lb      ci95ub            p treated comparator
#> 1 1.2399630 1.10356958   1.3923415 2.864684e-04  460047    1176571
#> 2 1.0722885 0.77165096   1.4812513 6.748151e-01  462499    1180683
#> 3 2.1434012 1.57458125   2.9235177 1.368282e-06  462535    1180974
#> 4 0.9612337 0.66877760   1.3728908 8.293856e-01  462643    1181034
#> 5 0.8564781 0.79784445   0.9189318 1.723412e-05  454020    1156084
#> 6 1.2756105 0.05932644  13.3125553 8.600838e-01  462836    1181468
#>   treatedDays comparatorDays eventsTreated eventsComparator        logRr
#> 1    46601026       74901764           560              726  0.21508151
#> 2    46902381       75196177            68               97  0.06979518
#> 3    46905353       75215995            98               79  0.76239393
```

```
#> 4    46914733        75217050              60              82 -0.03953771
#> 5    45951653        73563751            1265            2514 -0.15492657
#> 6    46941477        75252033               1               2  0.24342487
#>      seLogRr
#> 1 0.05929621
#> 2 0.16635769
#> 3 0.15785959
#> 4 0.18347845
#> 5 0.03604615
#> 6 1.38099677
```

This tells us, per drug-target(-indication)-outcome-analysis combination, the estimated relative risk and 95% confidence interval, as well as the number of people in the treated and comparator group (after trimming and matching if applicable), and the number of outcomes observed for those groups within the specified risk 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 outcomes, and the yellow diamond represents our health outcome of interest: GI bleed. An unbiased, well-calibrated analysis should have 95% of the negative controls between the dashed lines (ie. 95% should have p > .05).

```r
install.packages("EmpiricalCalibration")
library(EmpiricalCalibration)

# Analysis 1: No matching, simple outcome model
negCons <- analysisSum[analysisSum$analysisId == 1 & analysisSum$outcomeId != 192671, ]
hoi <-  analysisSum[analysisSum$analysisId == 1 & analysisSum$outcomeId == 192671, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(negCons$logRr, negCons$seLogRr, hoi$logRr, hoi$seLogRr, null)
```
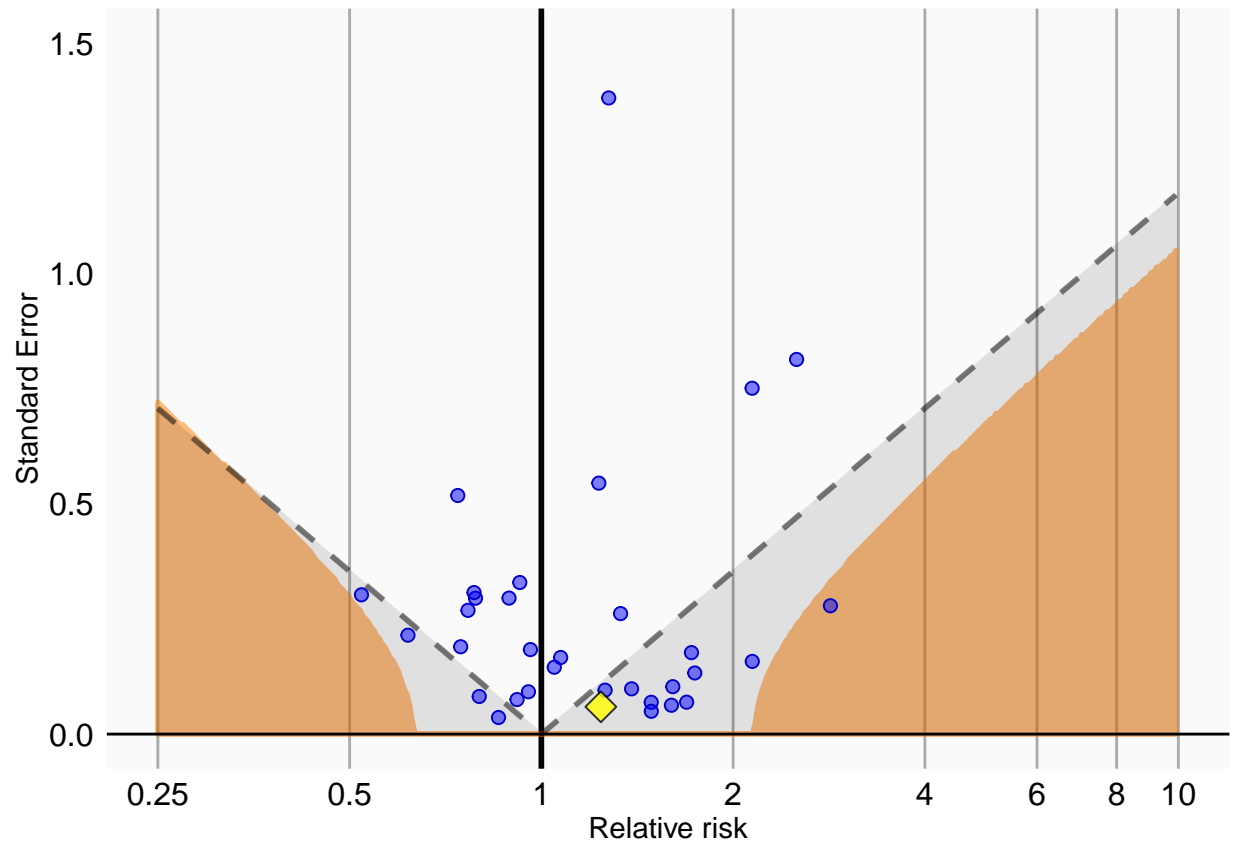
```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with NA
#> standard error detected. Removing before fitting null distribution
```
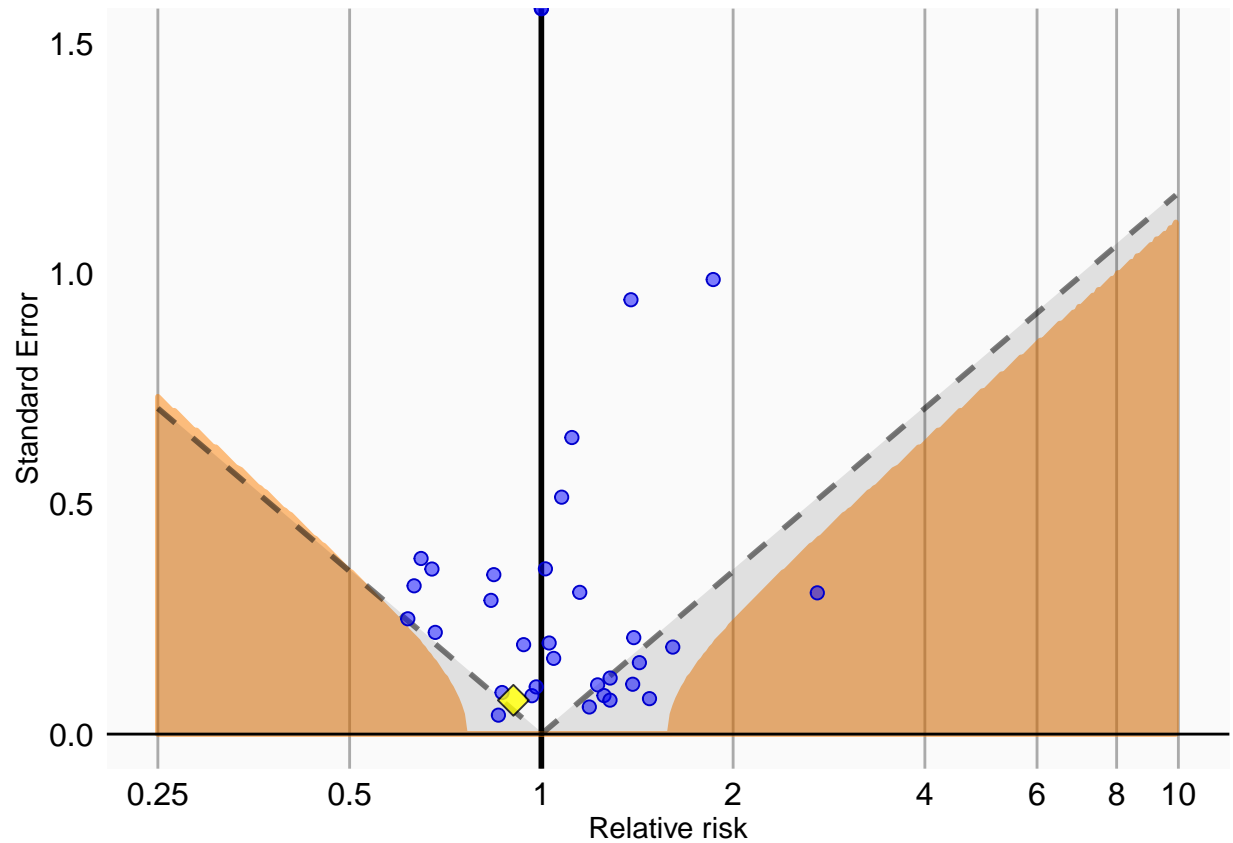
```
#> Warning: Removed 3 rows containing missing values (geom_point).
```

```
# Analysis 2: Matching plus simple outcome model
negCons <- analysisSum[analysisSum$analysisId == 2 & analysisSum$outcomeId != 192671, ]
hoi <-  analysisSum[analysisSum$analysisId == 2 & analysisSum$outcomeId == 192671, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(negCons$logRr, negCons$seLogRr, hoi$logRr, hoi$seLogRr, null)
```
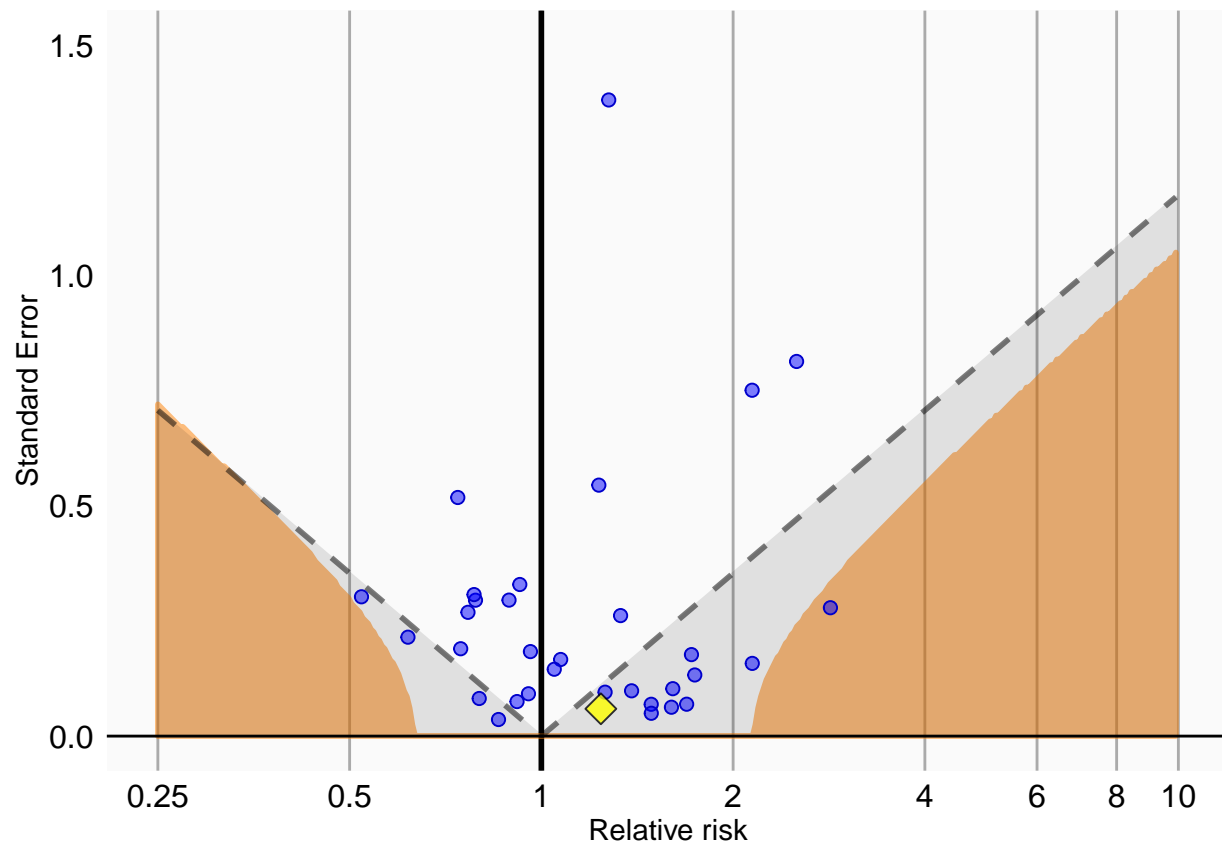
```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with
#> infinite standard error detected. Removing before fitting null distribution
```

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

```
#> Warning: Removed 2 rows containing missing values (geom_point).
```

```
# Analysis 3: Stratification plus simple outcome model
negCons <- analysisSum[analysisSum$analysisId == 3 & analysisSum$outcomeId != 192671, ]
hoi <-  analysisSum[analysisSum$analysisId == 3 & analysisSum$outcomeId == 192671, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(negCons$logRr, negCons$seLogRr, hoi$logRr, hoi$seLogRr, null)
```

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

```
#> Warning: Removed 3 rows containing missing values (geom_point).
```

```
# Analysis 4: Matching plus stratified outcome model
negCons <- analysisSum[analysisSum$analysisId == 4 & analysisSum$outcomeId != 192671, ]
hoi <-  analysisSum[analysisSum$analysisId == 4 & analysisSum$outcomeId == 192671, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(negCons$logRr, negCons$seLogRr, hoi$logRr, hoi$seLogRr, null)
```

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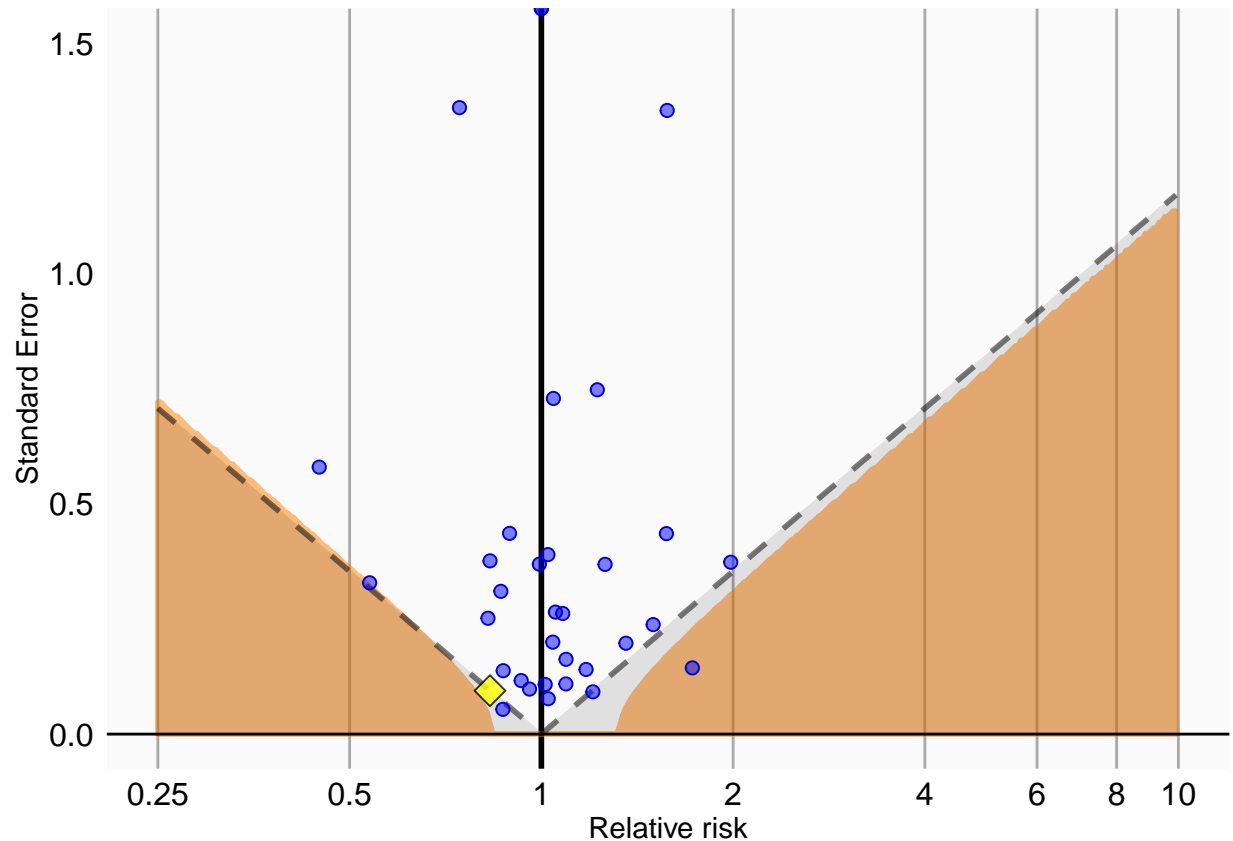
```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with NA
#> standard error detected. Removing before fitting null distribution
```

```
#> Warning: Removed 2 rows containing missing values (geom_point).
```

```
# Analysis 5: Matching plus full outcome model
negCons <- analysisSum[analysisSum$analysisId == 5 & analysisSum$outcomeId != 192671, ]
hoi <-  analysisSum[analysisSum$analysisId == 5 & analysisSum$outcomeId == 192671, ]
null <- fitNull(negCons$logRr, negCons$seLogRr)
plotCalibrationEffect(negCons$logRr, negCons$seLogRr, hoi$logRr, hoi$seLogRr, null)
```
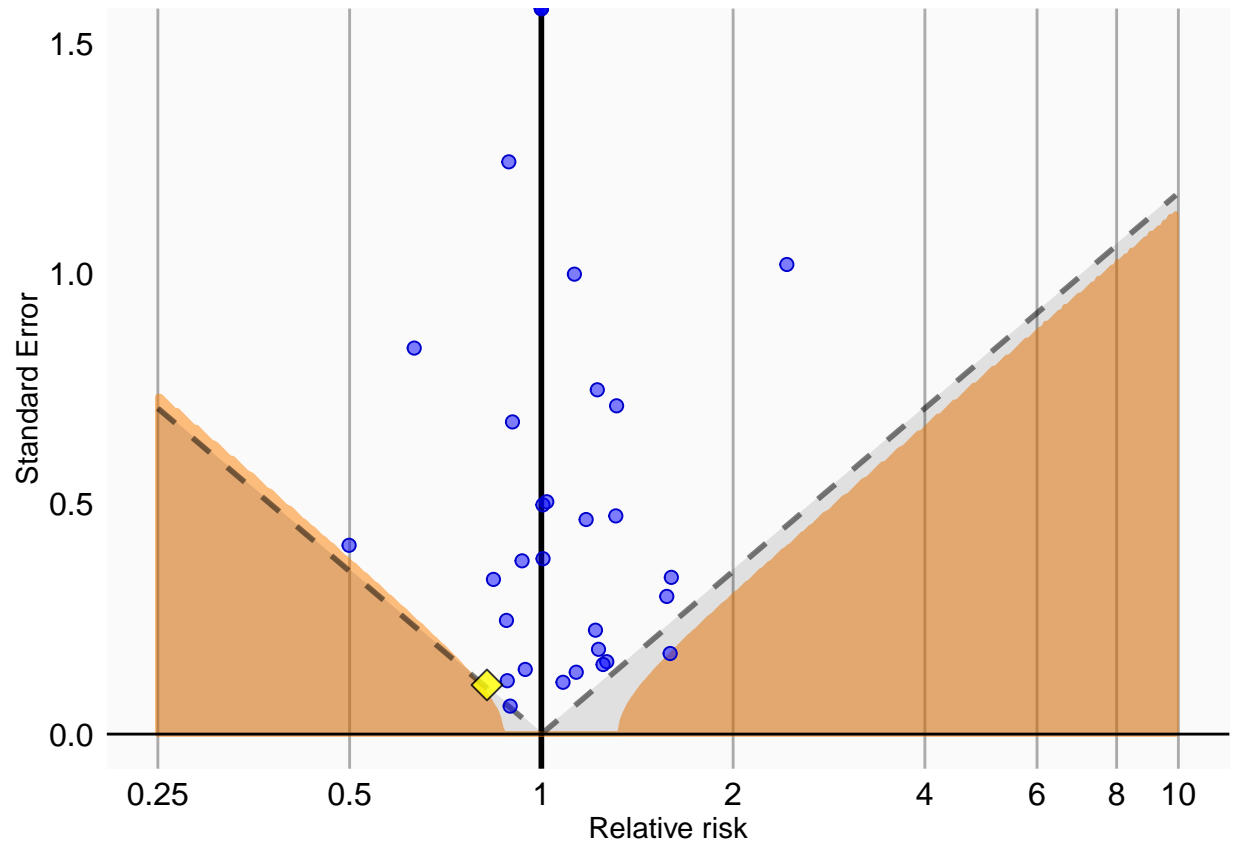
```
#> Warning in fitNull(negCons$logRr, negCons$seLogRr): Estimate(s) with
#> infinite standard error detected. Removing before fitting null distribution

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

#> Warning: Removed 3 rows containing missing values (geom_point).
```

8 Acknowledgments

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

```
citation("CohortMethod")
```

```
#>
#> To cite package 'CohortMethod' in publications use:
#>
#>   Martijn J. Schuemie, Marc A. Suchard and Patrick B. Ryan (2015).
#>   CohortMethod: New-user cohort method with large scale propensity
#>   and outcome models. R package version 1.1.3.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {CohortMethod: New-user cohort method with large scale propensity and outcome models},
#>     author = {Martijn J. Schuemie and Marc A. Suchard and Patrick B. Ryan},
#>     year = {2015},
#>     note = {R package version 1.1.3},
#>   }
#>
#> ATTENTION: This citation information has been auto-generated from
#> the package DESCRIPTION file and may need manual editing, see
```

```
#> 'help("citation")'.
```

Further, `CohortMethod` 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},
#>   }
```