Single studies using the CohortMethod package

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

2023-12-21

Contents

1 Introduction **1**

2 Installation instructions **2**

3 Data extraction **2**
3.1 Configuring the connection to the server . . . 2
3.2 Preparing the exposures and outcome(s) . . . 2
3.3 Extracting the data from the server . . . 4

4 Defining the study population **5**

5 Propensity scores **6**
5.1 Fitting a propensity model . . . 6
5.2 Propensity score diagnostics . . . 6
5.3 Using the propensity score . . . 7
5.4 Evaluating covariate balance . . . 7
5.5 Inspecting select population characteristics . . . 8
5.6 Generalizability . . . 8

6 Follow-up and power **8**

7 Outcome models **9**
7.1 Fitting a simple outcome model . . . 9
7.2 Adding interaction terms . . . 9
7.3 Adding covariates to the outcome model . . . 10
7.4 Inspecting the outcome model . . . 10
7.5 Kaplan-Meier plot . . . 10
7.6 Time-to-event plot . . . 10

8 Acknowledgments **11**

1 Introduction

This vignette describes how you can use the `CohortMethod` package to perform a single new-user cohort study. We will walk through all the steps needed to perform an exemplar study, and we have selected the well-studied topic of the effect of coxibs versus non-selective non-steroidal anti-inflammatory drugs (NSAIDs) on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one coxib – celecoxib – and one non-selective NSAID – diclofenac.

2 Installation instructions

Before installing the CohortMethod package make sure you have Java available. For Windows users, RTools is also necessary. See these instructions for properly configuring your R environment.

The CohortMethod package is currently maintained in a Github repository, and has dependencies on other packages in Github. All of these packages can be downloaded and installed from within R using the drat package:

```
install.packages("remotes")
remotes::install_github("ohdsi/CohortMethod")
```

Once installed, you can type library(CohortMethod) to load the package.

3 Data extraction

The first step in running the CohortMethod is extracting all necessary data from the database server holding the data in the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) format.

3.1 Configuring the connection to the server

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"
cohortDatabaseSchema <- "my_results"
cohortTable <- "my_cohorts"
options(sqlRenderTempEmulationSchema = NULL)
```

The last few lines define the cdmDatabaseSchema, cohortDatabaseSchema, and cohortTable variables. We'll use these later to tell R where the data in CDM format live, and where we want to write intermediate tables. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example cdmDatabaseSchema <- "my_cdm_data.dbo". For database platforms that do not support temp tables, such as Oracle, it is also necessary to provide a schema where the user has write access that can be used to emulate temp tables. PostgreSQL supports temp tables, so we can set options(sqlRenderTempEmulationSchema = NULL) (or not set the sqlRenderTempEmulationSchema at all.)

3.2 Preparing the exposures and outcome(s)

We need to define the exposures and outcomes for our study. Here, we will define our exposures using the OHDSI Capr package. We define two cohorts, one for celecoxib and one for diclofenac. For each cohort we require a prior diagnosis of 'osteoarthritis of knee', and 365 days of continuous prior observation. we restrict to the first exposure per person:

```
library(Capr)

osteoArthritisOfKneeConceptId <- 4079750
celecoxibConceptId <- 1118084
```

```
diclofenacConceptId <- 1124300
osteoArthritisOfKnee <- cs(
  descendants(osteoArthritisOfKneeConceptId),
  name = "Osteoarthritis of knee"
)
attrition = attrition(
  "prior osteoarthritis of knee" = withAll(
    atLeast(1, condition(osteoArthritisOfKnee), duringInterval(eventStarts(-Inf, 0)))
  )
)
celecoxib <- cs(
  descendants(celecoxibConceptId),
  name = "Celecoxib"
)
diclofenac  <- cs(
  descendants(diclofenacConceptId),
  name = "Diclofenac"
)
celecoxibCohort <- cohort(
  entry = entry(
    drug(celecoxib, firstOccurrence()),
    observationWindow = continuousObservation(priorDays = 365)
  ),
  attrition = attrition,
  exit = exit(endStrategy = drugExit(celecoxib,
                                     persistenceWindow = 30,
                                     surveillanceWindow = 0))
)
diclofenacCohort <- cohort(
  entry = entry(
    drug(diclofenac, firstOccurrence()),
    observationWindow = continuousObservation(priorDays = 365)
  ),
  attrition = attrition,
  exit = exit(endStrategy = drugExit(diclofenac,
                                     persistenceWindow = 30,
                                     surveillanceWindow = 0))
)
```

We'll pull the outcome definition from the OHDSI `PhenotypeLibrary`:

```
library(PhenotypeLibrary)
outcomeCohorts <- getPlCohortDefinitionSet(77) # GI bleed
```

We combine the exposure and outcome cohort definitions, and use `CohortGenerator` to generate the cohorts:

```
library(CirceR)
# For exposures, create a cohort definition set table as required by CohortGenerator:
exposureCohorts <- tibble(cohortId = c(1,2),
                          cohortName = c("Celecoxib", "Diclofenac"),
                          json = c(as.json(celecoxibCohort),
                                   as.json(diclofenacCohort)))
exposureCohorts$sql <- sapply(exposureCohorts$json,
                              buildCohortQuery,
                              options = createGenerateOptions())
```

```
allCohorts <- bind_rows(outcomeCohorts,
                        exposureCohorts)

library(CohortGenerator)
cohortTableNames <- getCohortTableNames(cohortTable = cohortTable)
createCohortTables(connectionDetails = connectionDetails,
                   cohortDatabaseSchema = cohortDatabaseSchema,
                   cohortTableNames = cohortTableNames)
generateCohortSet(connectionDetails = connectionDetails,
                  cdmDatabaseSchema = cdmDatabaseSchema,
                  cohortDatabaseSchema = cohortDatabaseSchema,
                  cohortTableNames = cohortTableNames,
                  cohortDefinitionSet = allCohorts)
```

If all went well, we now have a table with the cohorts of interest. We can see how many entries per cohort:

```
connection <- DatabaseConnector::connect(connectionDetails)
sql <- "SELECT cohort_definition_id, COUNT(*) AS count FROM @cohortDatabaseSchema.@cohortTable GROUP BY
DatabaseConnector::renderTranslateQuerySql(connection, sql,  cohortDatabaseSchema = cohortDatabaseSchema
DatabaseConnector::disconnect(connection)
```

```
##   cohort_concept_id  count
## 1                 1 109307
## 2                 2 176675
## 3                77 733601
```

3.3 Extracting the data from the server

Now we can tell `CohortMethod` to extract the cohorts, construct covariates, and extract all necessary data for our analysis.

Important: The target and comparator drug must not be included in the covariates, including any descendant concepts. You will need to manually add the drugs and descendants to the `excludedCovariateConceptIds` of the covariate settings. In this example code we exclude the concepts for celecoxib and diclofenac and specify `addDescendantsToExclude = TRUE`:

```
# Define which types of covariates must be constructed:
covSettings <- createDefaultCovariateSettings(
  excludedCovariateConceptIds = c(diclofenacConceptId, celecoxibConceptId),
  addDescendantsToExclude = TRUE
)

#Load data:
cohortMethodData <- getDbCohortMethodData(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  targetId = 1,
  comparatorId = 2,
  outcomeIds = 77,
  exposureDatabaseSchema = cohortDatabaseSchema,
  exposureTable = cohortTable,
  outcomeDatabaseSchema = cohortDatabaseSchema,
  outcomeTable = cohortTable,
  covariateSettings = covSettings
)
cohortMethodData
```

There are many parameters, but they are all documented in the CohortMethod manual. The createDefaultCovariateSettings function is described in the FeatureExtraction package. In short, we are pointing the function to the table created earlier and indicating which concept IDs in that table identify the target, comparator and outcome. We instruct that the default set of covariates should be constructed, including covariates for all conditions, drug exposures, and procedures that were found on or before the index date. To customize the set of covariates, please refer to the FeatureExtraction package vignette by typing vignette("UsingFeatureExtraction", package="FeatureExtraction").

All data about the cohorts, outcomes, and covariates are extracted from the server and stored in the cohortMethodData object. This object uses the Andromeda package to store information in a way that ensures R does not run out of memory, even when the data are large. We can use the generic summary() function to view some more information of the data we extracted:

```
summary(cohortMethodData)
```

3.3.1 Saving the data to file

Creating the cohortMethodData file can take considerable computing time, and it is probably a good idea to save it for future sessions. Because cohortMethodData uses Andromeda, we cannot use R's regular save function. Instead, we'll have to use the saveCohortMethodData() function:

```
saveCohortMethodData(cohortMethodData, "coxibVsNonselVsGiBleed.zip")
```

We can use the loadCohortMethodData() function to load the data in a future session.

3.3.2 Defining new users

Typically, a new user is defined as first time use of a drug (either target or comparator), and typically a washout period (a minimum number of days prior first use) is used to make sure it is truly first use. When using the CohortMethod package, you can enforce the necessary requirements for new use in three ways:

1. When creating the cohorts in the database, for example using Capr.
2. When loading the cohorts using the getDbCohortMethodData function, you can use the firstExposureOnly, removeDuplicateSubjects, restrictToCommonPeriod, and washoutPeriod arguments. (As shown in the example above).
3. When defining the study population using the createStudyPopulation function (see below) using the firstExposureOnly, removeDuplicateSubjects, restrictToCommonPeriod, and washoutPeriod arguments.

The advantage of option 1 is that the input cohorts are already fully defined outside of the CohortMethod package, and for example external cohort characterization tools can be used on the same cohorts used in this package. The advantage of options 2 and 3 is that it saves you the trouble of limiting to first use yourself, for example allowing you to directly use the drug_era table in the CDM. Option 2 is more efficient than 3, since only data for first use will be fetched, while option 3 is less efficient but allows you to compare the original cohorts to the study population.

4 Defining the study population

Typically, the exposure cohorts and outcome cohorts will be defined independently of each other. When we want to produce an effect size estimate, we need to further restrict these cohorts and put them together, for example by removing exposed subjects that had the outcome prior to exposure, and only keeping outcomes that fall within a defined risk window. For this we can use the createStudyPopulation function:

```
studyPop <- createStudyPopulation(
  cohortMethodData = cohortMethodData,
  outcomeId = 3,
  firstExposureOnly = FALSE,
```

```
  restrictToCommonPeriod = FALSE,
  washoutPeriod = 0,
  removeDuplicateSubjects = "keep all",
  removeSubjectsWithPriorOutcome = TRUE,
  minDaysAtRisk = 1,
  riskWindowStart = 0,
  startAnchor = "cohort start",
  riskWindowEnd = 30,
  endAnchor = "cohort end"
)
```

Note that we've set `firstExposureOnly` and `removeDuplicateSubjects` to FALSE, and `washoutPeriod` to zero because we already filtered on these arguments when using the `getDbCohortMethodData` function. During loading we set `restrictToCommonPeriod` to FALSE, and we do the same here because we do not want to force the comparison to restrict only to time when both drugs are recorded. We specify the outcome ID we will use, and that people with outcomes prior to the risk window start date will be removed. The risk window is defined as starting at the cohort start date (the index date, `riskWindowStart = 0` and `startAnchor = "cohort start"`), and the risk windows ends 30 days after the cohort ends (`riskWindowEnd = 30` and `endAnchor = "cohort end"`). Note that the risk windows are truncated at the end of observation or the study end date. We also remove subjects who have no time at risk. To see how many people are left in the study population we can always use the `getAttritionTable` function:

```
getAttritionTable(studyPop)
```

One additional filtering step that is often used is matching or trimming on propensity scores, as will be discussed next.

5 Propensity scores

The `CohortMethod` can use propensity scores to adjust for potential confounders. Instead of the traditional approach of using a handful of predefined covariates, `CohortMethod` typically uses thousands to millions of covariates that are automatically constructed based on conditions, procedures and drugs in the records of the subjects.

5.1 Fitting a propensity model

We can fit a propensity model using the covariates constructed by the `getDbcohortMethodData()` function:

```
ps <- createPs(cohortMethodData = cohortMethodData, population = studyPop)
```

The `createPs()` function uses the `Cyclops` package to fit a large-scale regularized logistic regression.

To fit the propensity model, `Cyclops` needs to know the hyperparameter value which specifies the variance of the prior. By default `Cyclops` will use cross-validation to estimate the optimal hyperparameter. However, be aware that this can take a really long time. You can use the `prior` and `control` parameters of the `createPs()` to specify `Cyclops` behavior, including using multiple CPUs to speed-up the cross-validation.

5.2 Propensity score diagnostics

We can compute the area under the receiver-operator curve (AUC) for the propensity score model:

```
computePsAuc(ps)
```

We can also plot the propensity score distribution, although we prefer the preference score distribution:

```
plotPs(ps,
       scale = "preference",
```

```
        showCountsLabel = TRUE,
        showAucLabel = TRUE,
        showEquiposeLabel = TRUE)
```

It is also possible to inspect the propensity model itself by showing the covariates that have non-zero coefficients:

```
getPsModel(ps, cohortMethodData)
```

One advantage of using the regularization when fitting the propensity model is that most coefficients will shrink to zero and fall out of the model. It is a good idea to inspect the remaining variables for anything that should not be there, for example variations of the drugs of interest that we forgot to exclude.

Finally, we can inspect the percent of the population in equipoise, meaning they have a prefence score between 0.3 and 0.7:

```
CohortMethod::computeEquipoise(ps)
```

A low equipoise indicates there is little overlap between the target and comparator populations.

5.3 Using the propensity score

We can use the propensity scores to trim, stratify, match, or weigh our population. For example, one could trim to equipoise, meaning only subjects with a preference score between 0.25 and 0.75 are kept:

```
trimmedPop <- trimByPsToEquipoise(ps)
plotPs(trimmedPop, ps, scale = "preference")
```

Instead (or additionally), we could stratify the population based on the propensity score:

```
stratifiedPop <- stratifyByPs(ps, numberOfStrata = 5)
plotPs(stratifiedPop, ps, scale = "preference")
```

We can also match subjects based on propensity scores. In this example, we're using one-to-one matching:

```
  matchedPop <- matchOnPs(ps, caliper = 0.2, caliperScale = "standardized logit", maxRatio = 1)
  plotPs(matchedPop, ps)
```

Note that for both stratification and matching it is possible to specify additional matching criteria such as age and sex using the `stratifyByPsAndCovariates()` and `matchOnPsAndCovariates()` functions, respectively.

We can see the effect of trimming and/or matching on the population using the `getAttritionTable` function:

```
getAttritionTable(matchedPop)
```

Or, if we like, we can plot an attrition diagram:

```
drawAttritionDiagram(matchedPop)
```

5.4 Evaluating covariate balance

To evaluate whether our use of the propensity score is indeed making the two cohorts more comparable, we can compute the covariate balance before and after trimming, matching, and/or stratifying:

```
balance <- computeCovariateBalance(matchedPop, cohortMethodData)
```

```
plotCovariateBalanceScatterPlot(balance, showCovariateCountLabel = TRUE, showMaxLabel = TRUE)
```

```
plotCovariateBalanceOfTopVariables(balance)
```

The 'before matching' population is the population as extracted by the `getDbCohortMethodData` function, so before any further filtering steps.

5.5 Inspecting select population characteristics

It is customary to include a table in your paper that lists some select population characteristics before and after matching/stratification/trimming. This is usually the first table, and so will be referred to as 'table 1'. To generate this table, you can use the `createCmTable1` function:

```
createCmTable1(balance)
```

5.6 Generalizability

The goal of any propensity score adjustments is typically to make the target and comparator cohorts comparably, to allow proper causal inference. However, in doing so, we often need to modify our population, for example dropping subjects that have no counterpart in the other exposure cohort. The population we end up estimating an effect for may end up being very different from the population we started with. An important question is: how different? And it what ways? If the populations before and after adjustment are very different, our estimated effect may not generalize to the original population (if effect modification is present). The `getGeneralizabilityTable()` function informs on these differences:

```
getGeneralizabilityTable(balance)
```

In this case, because we used PS matching, we are likely aiming to estimate the average treatment effect in the treated (ATT). For this reason, the `getGeneralizabilityTable()` function automatically selected the target cohort as the basis for evaluating generalizability: it shows, for each covariate, the mean value before and PS adjustment in the target cohort. Also shown is the standardized difference of mean, and the table is reverse sorted by the absolute standard difference of mean (ASDM).

6 Follow-up and power

Before we start fitting an outcome model, we might be interested to know whether we have sufficient power to detect a particular effect size. It makes sense to perform these power calculations once the study population has been fully defined, so taking into account loss to the various inclusion and exclusion criteria (such as no prior outcomes), and loss due to matching and/or trimming. Since the sample size is fixed in retrospective studies (the data has already been collected), and the true effect size is unknown, the CohortMethod package provides a function to compute the minimum detectable relative risk (MDRR) instead:

```
computeMdrr(
  population = studyPop,
  modelType = "cox",
  alpha = 0.05,
  power = 0.8,
  twoSided = TRUE
)
```

In this example we used the `studyPop` object, so the population before any matching or trimming. If we want to know the MDRR after matching, we use the `matchedPop` object we created earlier instead:

```
computeMdrr(
  population = matchedPop,
  modelType = "cox",
  alpha = 0.05,
  power = 0.8,
  twoSided = TRUE
)
```

Even thought the MDRR in the matched population is higher, meaning we have less power, we should of course not be fooled: matching most likely eliminates confounding, and is therefore preferred to not matching.

To gain a better understanding of the amount of follow-up available we can also inspect the distribution of follow-up time. We defined follow-up time as time at risk, so not censored by the occurrence of the outcome. The `getFollowUpDistribution` can provide a simple overview:

```
getFollowUpDistribution(population = matchedPop)
```

The output is telling us number of days of follow-up each quantile of the study population has. We can also plot the distribution:

```
plotFollowUpDistribution(population = matchedPop)
```

7 Outcome models

The outcome model is a model describing which variables are associated with the outcome.

7.1 Fitting a simple outcome model

In theory we could fit an outcome model without using the propensity scores. In this example we are fitting an outcome model using a Cox regression:

```
outcomeModel <- fitOutcomeModel(population = studyPop,
                                modelType = "cox")
outcomeModel
```

But of course we want to make use of the matching done on the propensity score:

```
outcomeModel <- fitOutcomeModel(population = matchedPop,
                                modelType = "cox",
                                stratified = TRUE)
outcomeModel
```

Note that we define the sub-population to be only those in the `matchedPop` object, which we created earlier by matching on the propensity score. We also now use a stratified Cox model, conditioning on the propensity score match sets.

Instead of matching or stratifying we can also perform Inverse Probability of Treatment Weighting (IPTW):

```
outcomeModel <- fitOutcomeModel(population = ps,
                                modelType = "cox",
                                inversePtWeighting = TRUE)
outcomeModel
```

7.2 Adding interaction terms

We may be interested whether the effect is different across different groups in the population. To explore this, we may include interaction terms in the model. In this example we include three interaction terms:

```
interactionCovariateIds <- c(8532001, 201826210, 21600960413)
# 8532001 = Female
# 201826210 = Type 2 Diabetes
# 21600960413 = Concurent use of antithrombotic agents
outcomeModel <- fitOutcomeModel(population = matchedPop,
                                modelType = "cox",
                                stratified = TRUE,
```

```
                                    interactionCovariateIds = interactionCovariateIds)
outcomeModel
```

Note that you can use the `grepCovariateNames` to find covariate IDs.

It is prudent to verify that covariate balance has also been achieved in the subgroups of interest. For example, we can check the covariate balance in the subpopulation of females:

```
balanceFemale <- computeCovariateBalance(population = matchedPop,
                                         cohortMethodData = cohortMethodData,
                                         subgroupCovariateId = 8532001)
plotCovariateBalanceScatterPlot(balanceFemale)
```

7.3 Adding covariates to the outcome model

One final refinement would be to use the same covariates we used to fit the propensity model to also fit the outcome model. This way we are more robust against misspecification of the model, and more likely to remove bias. For this we use the regularized Cox regression in the `Cyclops` package. (Note that the treatment variable is automatically excluded from regularization.)

```
outcomeModel <- fitOutcomeModel(population = matchedPop,
                                cohortMethodData = cohortMethodData,
                                modelType = "cox",
                                stratified = TRUE,
                                useCovariates = TRUE)
outcomeModel
```

7.4 Inspecting the outcome model

We can inspect more details of the outcome model:

```
exp(coef(outcomeModel))
```

```
exp(confint(outcomeModel))
```

We can also see the covariates that ended up in the outcome model:

```
getOutcomeModel(outcomeModel, cohortMethodData)
```

7.5 Kaplan-Meier plot

We can create the Kaplan-Meier plot:

```
plotKaplanMeier(matchedPop, includeZero = FALSE)
```

Note that the Kaplan-Meier plot will automatically adjust for any stratification, matching, or trimming that may have been applied.

7.6 Time-to-event plot

We can also plot time-to-event, showing both events before and after the index date, and events during and outside the defined time-at-risk window. This plot can provide insight into the temporal pattern of the outcome relative to the exposures:

```
plotTimeToEvent(cohortMethodData = cohortMethodData,
                outcomeId = 77,
                firstExposureOnly = FALSE,
                washoutPeriod = 0,
```

```
                removeDuplicateSubjects = "keep all",
                minDaysAtRisk = 1,
                riskWindowStart = 0,
                startAnchor = "cohort start",
                riskWindowEnd = 30,
                endAnchor = "cohort end")
```

Note that this plot does not show any adjustment for the propensity score.

8 Acknowledgments

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

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

```
##
## To cite package 'CohortMethod' in publications use:
##
##   Schuemie M, Suchard M, Ryan P (2023). _CohortMethod: New-User Cohort Method with Large Scale Propen
##   https://github.com/OHDSI/CohortMethod.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {CohortMethod: New-User Cohort Method with Large Scale Propensity and Outcome Models},
##     author = {Martijn Schuemie and Marc Suchard and Patrick Ryan},
##     year = {2023},
##     note = {https://ohdsi.github.io/CohortMethod, https://github.com/OHDSI/CohortMethod},
##   }
```

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, Madigan D (2013). "Massive parallelization of serial infe
##   Modeling and Computer Simulation_, *23*, 10. <https://dl.acm.org/doi/10.1145/2414416.2414791>.
##
## 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 mo
##     journal = {ACM Transactions on Modeling and Computer Simulation},
##     volume = {23},
##     pages = {10},
##     year = {2013},
##     url = {https://dl.acm.org/doi/10.1145/2414416.2414791},
##   }
```

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