Single studies using the CohortMethod package

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

2016-08-08

Contents

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. Java can be downloaded from www.java.com. For Windows users, RTools is also necessary. RTools can be downloaded from CRAN.

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 devtools package:

```
install.packages("devtools")
library(devtools)
install_github("ohdsi/OhdsiRTools")
install_github("ohdsi/SqlRender")
install_github("ohdsi/DatabaseConnector")
install_github("ohdsi/Cyclops")
install_github("ohdsi/PatientLevelPrediction")
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"
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".

3.2 Preparing the exposures and outcome(s)

We need to define the exposures and outcomes for our study. One could use an external cohort definition tools, but in this example we do this by writing SQL statements against the OMOP CDM that populate a table of events in which we are interested. The resulting table should have the same structure as the `cohort` table in the CDM. For CDM v5+, this means it should have the fields `cohort_definition_id`, `cohort_start_date`, `cohort_end_date`,and `subject_id`. For CDM v4, the `cohort_definition_id` field must be called `cohort_concept_id`.

For our example study, we have created a file called *coxibVsNonselVsGiBleed.sql* with the following contents:

```
/***********************************
File coxibVsNonselVsGiBleed.sql
***********************************/

IF OBJECT_ID('@resultsDatabaseSchema.coxibVsNonselVsGiBleed', 'U') IS NOT NULL
  DROP TABLE @resultsDatabaseSchema.coxibVsNonselVsGiBleed;

CREATE TABLE @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
  cohort_definition_id INT,
  cohort_start_date DATE,
    cohort_end_date DATE,
    subject_id BIGINT
    );

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 1, -- Exposure
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM @cdmDatabaseSchema.drug_era
WHERE drug_concept_id = 1118084;-- celecoxib

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 2, -- Comparator
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM @cdmDatabaseSchema.drug_era
WHERE drug_concept_id = 1124300; --diclofenac

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 3, -- Outcome
```

```
    condition_start_date,
    condition_end_date,
    condition_occurrence.person_id
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);
```

Note on 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.

```
library(SqlRender)
sql <- readSql("coxibVsNonselVsGiBleed.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.

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

```
sql <- paste("SELECT cohort_definition_id, COUNT(*) AS count",
             "FROM @resultsDatabaseSchema.coxibVsNonselVsGiBleed",
             "GROUP BY cohort_definition_id")
sql <- renderSql(sql, resultsDatabaseSchema = resultsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
#>   cohort_concept_id  count
#> 1                 1 129173
#> 2                 2 420205
#> 3                 3 422274
```

3.3 Extracting the data from the server

Now we can tell `CohortMethod` to define the cohorts based on our events, and extract all necessary data for our analysis:

```
# 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

# Define which types of covariates must be constructed:
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 = nsaids,
                                         deleteCovariatesSmallCount = 100)

#Load data:
cohortMethodData <- getDbCohortMethodData(connectionDetails = connectionDetails,
                                          cdmDatabaseSchema = cdmDatabaseSchema,
                                          oracleTempSchema = resultsDatabaseSchema,
                                          targetId = 1,
                                          comparatorId = 2,
                                          outcomeIds = 3,
                                          studyStartDate = "",
                                          studyEndDate = "",
                                          exposureDatabaseSchema = resultsDatabaseSchema,
                                          exposureTable = "coxibVsNonselVsGiBleed",
                                          outcomeDatabaseSchema = resultsDatabaseSchema,
                                          outcomeTable = "coxibVsNonselVsGiBleed",
                                          cdmVersion = cdmVersion,
                                          excludeDrugsFromCovariates = FALSE,
                                          firstExposureOnly = TRUE,
                                          removeDuplicateSubjects = TRUE,
                                          washoutPeriod = 180,
                                          covariateSettings = covariateSettings)
cohortMethodData
```

There are many parameters, but they are all documented in the `CohortMethod` manual. The `createCovariateSettings` 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 many different covariates should be constructed, including covariates for all conditions, drug exposures, and procedures that were found on or before the index date.

Important: The target and comparator drug must not be included in the covariates, including any descendant concepts. If the `targetId` and `comparatorId` arguments represent real concept IDs, you can set the `excludeDrugsFromCovariates` argument to TRUE and automatically the drugs and their descendants will be excluded from the covariates. However, if the `targetId` and `comparatorId` arguments do not represent concept IDs, such as in the example above, you will need to manually add the drugs and descendants to the `excludedCovariateConceptIds` of the`covariateSettings` argument.

All data about the cohorts, outcomes, and covariates are extracted from the server and stored in the `cohortMethodData` object. This object uses the package `ff` 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 `ff`, we cannot use R's regular save function. Instead, we'll have to use the `saveCohortMethodData()` function:

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

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 when using a cohort definition tool.
2. When loading the cohorts using the `getDbCohortMethodData` function, you can use the `firstExposureOnly`, `removeDuplicateSubjects`, 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`, 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,
                                  washoutPeriod = 0,
                                  removeDuplicateSubjects = FALSE,
                                  removeSubjectsWithPriorOutcome = TRUE,
                                  minDaysAtRisk = 1,
                                  riskWindowStart = 0,
                                  addExposureDaysToStart = FALSE,
                                  riskWindowEnd = 30,
                                  addExposureDaysToEnd = TRUE)
```

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. 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 index date (`riskWindowStart = 0` and `addExposureDaysToStart = FALSE`), and the risk windows ends 30 days after exposure ends (`riskWindowEnd = 30` and `addExposureDaysToEnd = TRUE`). 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:

```r
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:

```r
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:

```r
computePsAuc(ps)
```

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

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

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

```r
propensityModel <- getPsModel(ps, cohortMethodData)
head(propensityModel)
```

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.

5.3 Using the propensity score

We can use the propensity scores to trim, stratify, or match 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 = 0.25, caliperScale = "standardized", 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)
```

```
plotCovariateBalanceOfTopVariables(balance)
```

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

5.5 Inserting the population cohort in the database

For various reasons it might be necessary to insert the study population back into the database, for example because we want to use an external cohort characterization tool. We can use the `insertDbPopulation` function for this purpose:

```
insertDbPopulation(population = matchedPop,
                   cohortIds = c(101,100),
                   connectionDetails = connectionDetails,
                   cohortDatabaseSchema = resultsDatabaseSchema,
                   cohortTable = "coxibVsNonselVsGiBleed",
                   createTable = FALSE,
                   cdmVersion = cdmVersion)
```

This function will store the population in a table with the same structure as the `cohort` table in the CDM, in this case in the same table where we had created our original cohorts.

6 Outcome models

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

6.1 Fitting the 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",
                                stratified = FALSE,
                                useCovariates = FALSE)
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,
                                useCovariates = FALSE)
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.

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
```

6.2 Inpecting the outcome model

We can inspect more details of the outcome model:

```
summary(outcomeModel)
```

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

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

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

```
fullOutcomeModel <- getOutcomeModel(outcomeModel, cohortMethodData)
head(fullOutcomeModel)
```

6.3 Kaplan-Meier plot

We can create the Kaplan-Meier plot:

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

7 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 2.0.3.
#>
#> 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 2.0.3},
#>   }
#>
#> 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.

```r
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.