Single studies using the CaseControl package

Martijn J. Schuemie

2016-06-29

Contents

1 Introduction

This vignette describes how you can use the `CaseControl` package to perform a single case-control 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 NSAIDs on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one NSAID: diclofenac.

2 Installation instructions

Before installing the `CaseControl` 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 `CaseControl` 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/CaseControl")
```

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

3 Overview

In the `CaseControl` package a study requires at least five steps:

1. Loading data on the cases and potential controls from the database needed for matching.
2. Selecting controls per case.
3. Loading exposure information for cases and controls.
4. Determining exposure status for cases and controls based on a definition of the risk window.
5. Fitting the model using conditional logistic regression.

In the following sections these steps will be demonstrated.

4 Configuring the connection to the server

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

The last three lines define the `cdmDatabaseSchema` and `cohortDatabaseSchema` variables,as well as the CDM version. We'll use these later to tell R where the data in CDM format live, where we have stored our cohorts of interest, 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"`.

5 Preparing the health outcome of interest and nesting cohort

We need to define the exposures and outcomes for our study. Additionally, we can specify a cohort in which to nest the study. The CDM also already contains standard cohorts in the `drug_era` and `condition_era`

table that could be used if those meet the requirement of the study, but often we require custom cohort definitions. One way to define cohorts is 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, meaning it should have the fields `cohort_definition_id`, `cohort_start_date`, `cohort_end_date`,and `subject_id`.

For our example study, we will rely on `drug_era` to define exposures, and we have created a file called *vignette.sql* with the following contents to define the outcome and the nesting cohort:

```
/**********************************
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 cohort 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 three 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 outcome of interest and the nesting cohort. We can see how many events:

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

querySql(connection, sql)
```

```
#>   cohort_definition_id  count
#> 1                    1 422274
#> 2                    2 118430
```

6 Extracting the data from the server

Now we can tell `CaseControl` to extract the necessary data on cases and potential controls:

```
caseData <- getDbCaseData(connectionDetails = connectionDetails,
                          cdmDatabaseSchema = cdmDatabaseSchema,
                          oracleTempSchema = oracleTempSchema,
                          outcomeDatabaseSchema = cohortDatabaseSchema,
                          cohortTable = cohortTable,
                          outcomeIds = 1,
                          useNestingCohort = TRUE,
                          nestingCohortDatabaseSchema = cohortDatabaseSchema,
                          nestingCohortTable = cohortTable,
                          nestingCohortId = 2,
                          useObservationEndAsNestingEndDate = TRUE,
                          getVisits = TRUE)
caseData
```

```
#> Case data object
#>
#> Outcome concept ID(s): 1
#> Nesting cohort ID: 2
```

There are many parameters, but they are all documented in the `CaseControl` manual. In short, we are pointing the function to the table created earlier and indicating which concept ID in that table identifies the outcome. Note that it is possible to fetch the data for multiple outcomes at once for efficiency. We furthermore specify a nesting cohort in the same table, meaning that people will be eligible to be cases or controls if and when they fall inside the specified cohort. In this case, the nesting cohort starts when people have their first diagnosis of rheumatoid arthritis. We use the `useObservationEndAsNestingEndDate` argument to indicate people will stay eligible until the end of their observation period. We furthermore specify we want to retrieve data on patient visits, which will be used later on for matching.

Data about the cases and potential controls are extracted from the server and stored in the `caseData` 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(caseData)
```

```
#> caseData object summary
#>
#> Outcome concept ID(s): 1
#> Nesting cohort ID: 2
#>
#> Population count: 118432
#> Population window count: 118432
#>
#> Outcome counts:
#>   Event count Case count
#> 1        9346       6422
```

6.1 Saving the data to file

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

```
saveCaseData(caseData, "GiBleed")
```

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

7 Selecting controls

Next, we can use the data to select matched controls per case:

```
caseControls <- selectControls(caseData = caseData,
                               outcomeId = 1,
                               firstOutcomeOnly = TRUE,
                               washoutPeriod = 180,
                               controlsPerCase = 2,
                               matchOnAge = TRUE,
                               ageCaliper = 2,
                               matchOnGender = TRUE,
```

```
                                    matchOnProvider = FALSE,
                                    matchOnVisitDate = TRUE,
                                    visitDateCaliper = 30,
                                    removedUnmatchedCases = TRUE)
```

In this example, we specify a washout period of 180 days, meaning that both cases and controls are required to have a minimum of 180 days of observation prior to the index date. We also specify we will only consider the first outcome per person. If a person's first outcome is within the washout period, that person will be removed from the analysis. We match on calendar time (implicit), age, gender, and visit date, and randomly sample up to two controls per case. Matching on visit date implies that the control should have a visit within n days from the index date of the case (n = 30 in this example), and the index date for the control will be set to the visit date. The purpose of this is to make the index dates more similar between cases and control, because almost by definition we would expect the case to have a visit on the index date. Finally, we ask that cases for which no matching control was found be removed from the analysis.

The `caseControls` object is a data frame with four columns:

```
head(caseControls)
```

```
#>   personId  indexDate isCase stratumId
#> 1        3 2009-10-10   TRUE         1
#> 2      123 2009-10-11  FALSE         1
#> 3      345 2009-10-09  FALSE         1
#> 4        6 2010-05-04   TRUE         2
#> 5      234 2010-05-04  FALSE         2
#> 6      567 2010-05-05  FALSE         2
```

We can show the attrition to see why cases and events were filtered:

```
getAttritionTable(caseControls)
```

```
#>                       description eventCount caseCount
#> 1                Original counts       9346      6422
#> 2               First event only       6422      6422
#> 3 Require 180 days of prior obs.       5815      5815
#> 4      Remove unmatched controls       5815      5815
```

8 Loading exposure data and covariates

Next, we load information on the exposures of the cases and controls, as well as covariates.

```
covarSettings <- createCovariateSettings(useCovariateRiskScores = TRUE,
                                         useCovariateRiskScoresCharlson = TRUE,
                                         useCovariateRiskScoresDCSI = TRUE,
                                         useCovariateRiskScoresCHADS2 = TRUE)

caseControlsExposure <- getDbExposureData(connectionDetails = connectionDetails,
                                          caseControls = caseControls,
                                          oracleTempSchema = oracleTempSchema,
                                          exposureDatabaseSchema = cdmDatabaseSchema,
                                          exposureTable = "drug_era",
```

```
                                                exposureIds = 1124300,
                                                covariateSettings = covarSettings)
```

Here we specify that we will use the `drug_era` table to identify exposures, and will only retrieve data on exposure to Diclofenac (concept ID 1124300). We use the `covariateSettings` function of the FeatureExtraction package to specify which covariates to construct. In this example the only coviariates are the Charlson comorbidity index, Diabetes Complications Severity Index (DCSI), and CHADS2 score.

When including covariates, the `caseControlsExposure` object uses `ff` to store large objects and therefore cannot be stored using standard functions. Please use `saveCaseControlsExposure` to save:

```
saveCaseControlsExposure(caseControlsExposure, "caseControlsExposure")
```

9 Creating case-control data

We can now use the exposure data to determine exposure status for the cases and controls. (The reason why fetching exposure data and determining exposure status is split up is that this is more efficient when we want to evaluate several risk window definitions.)

```
caseControlData <- createCaseControlData(caseControlsExposure = caseControlsExposure,
                                         exposureId = 1124300,
                                         firstExposureOnly = FALSE,
                                         riskWindowStart = 0,
                                         riskWindowEnd = 0)
```

Here we specify we are intested in all exposures, not just the first one, and that the exposure should overlap with the index date (the risk window starts and ends on day 0, the index date). The `caseControlData` object is a data frame with five columns:

```
head(caseControlData)
```

```
#>   personId  indexDate isCase stratumId rowId exposed
#> 1        3 2009-10-10   TRUE         1     1       1
#> 2      123 2009-10-11  FALSE         1     2       0
#> 3      345 2009-10-09  FALSE         1     3       0
#> 4        6 2010-05-04   TRUE         2     4       0
#> 5      234 2010-05-04  FALSE         2     5       0
#> 6      567 2010-05-05  FALSE         2     6       0
```

10 Fitting the model

We can now fit the model, which is a logistic regression conditioned on the matched sets:

```
fit <- fitCaseControlModel(caseControlData,
                           useCovariates = TRUE,
                           caseControlsExposure = caseControlsExposure,
                           prior = createPrior("none"))

fit
```

```
#> Case-Control fitted model
#> Status: OK
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment 1.103866  0.825774  1.464080 0.098819  0.1461
```

The generic functions `summary`, `coef`, and `confint` are implemented for the `fit` object:

```
summary(fit)
```

```
#> Case-Control fitted model
#> Status: OK
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment 1.103866  0.825774  1.464080 0.098819  0.1461
#>
#> Counts
#>       Cases Controls Exposed cases Exposed controls
#> Count  5815    11630            78               164
```

```
coef(fit)
```

```
#> [1] 0.0988189
```

```
confint(fit)
```

```
#> [1] -0.1914345  0.3812269
```

11 Acknowledgments

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

```
citation("CaseControl")
```

```
#>
#> To cite package 'CaseControl' in publications use:
#>
#>   Martijn Schuemie (2016). CaseControl: Case-Control. R package
#>   version 0.0.3.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {CaseControl: Case-Control},
#>     author = {Martijn Schuemie},
#>     year = {2016},
#>     note = {R package version 0.0.3},
#>   }
#>
#> ATTENTION: This citation information has been auto-generated from
#> the package DESCRIPTION file and may need manual editing, see
#> 'help("citation")'.
```

Furthermore, `CaseControl` makes 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},
#>   }
```