Running cohort diagnostics using WebAPI

Martijn J. Schuemie

2020-09-03

Contents

1 Introduction

This vignette describes how one could use the CohortDiagnostics R package, using the OHDSI WebAPI to access cohort definitions.

The CohortDiagnostics package allows one to generate a wide set of diagnostics to evaluate cohort definitions against a database in the Common Data Model (CDM). These diagnostics include incidence rates (optionally stratified by age, gender, and calendar year), cohort characteristics (comorbidities, drug use, etc.), and the codes found in the data triggering the various rules in the cohort definitions.

The CohortDiagnostics package in general works in two steps:

1. Generate the diagnostics against a database in the CDM.
2. Explore the generated diagnostics in a Shiny app included in the CohortDiagnostics package.

There are currently two approaches one can take to step 1: The cohort diagnostics can be embedded in an OHDSI study package, where all the cohort definitions are stored as part of that study package, or the cohort diagnostics can be used as a stand-alone solution, relying on a WebAPI instance to provide the cohort definitions. WebAPI is the backend of the OHDSI ATLAS application, allowing programmatic access to the cohort definitions created in ATLAS. This vignette describes the latter approach: how to run CohortDiagnostics using the WebAPI.

2 Running the cohort diagnostics

2.1 Defining the set of cohorts to diagnose

The first step is to define the set of cohorts we wish to create diagnostics for. We do this by creating a data frame with four columns:

- **atlasId**: The cohort ID in ATLAS.
- **atlasName**: The full name of the cohort. This will be shown in the Shiny app.
- **cohortId**: The cohort ID to use in the package. Usually the same as the cohort ID in ATLAS.
- **name**: A short name for the cohort, to use to create file names. do not use special characters.

A convenient way to create such a data frame is to create a CSV file, and load it into R. Here is an example table we assume is stored in `cohortsToDiagnose.csv`:

atlasId	atlasName	cohortId	name
1770710	New users of ACE inhibitors as first-line monotherapy for hypertension	1770710	ace_inhibitors
1770711	New users of Thiazide-like diuretics as first-line monotherapy for hypertension	1770711	thz
1770712	Angioedema outcome	1770712	angioedema
1770713	Acute myocardial infarction outcome	1770713	ami

We can read the table using

```
library(CohortDiagnostics)
cohortSetReference <- read.csv("cohortsToDiagnose.csv")
```

2.2 Configuring the connection to the server

We need to tell R how to connect to the server where the data are. `CohortDiagnostics` 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"
oracleTempSchema <- NULL
cohortDatabaseSchema <- "my_schema"
cohortTable <- "my_cohort_table"
```

The last four lines define the `cdmDatabaseSchema`, `oracleTempSchema`, `cohortDatabaseSchema`, and `cohortTable` variables. We'll use the `cdmDatabaseSchema` later to tell R where the data in CDM format

live. The `oracleTempSchema` is needed only for Oracle users, since Oracle does not support temporary tables. The `cohortDatabaseSchema`, and `cohortTable` specify where we want to instantiate our cohorts. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

2.3 Creating a new cohort table

In order to run most of the cohort diagnostics, we need to instantiate the cohorts. The best way is to instantiate the cohorts in a new cohort table. We can use the `createCohortTable` to create an empty cohort table:

```
createCohortTable(connectionDetails = connectionDetails,
                  cohortDatabaseSchema = cohortDatabaseSchema,
                  cohortTable = cohortTable)
```

Note this this function will **delete the table if it already exists** before creating it.

2.4 Instantiating the cohorts

Next, we will instantiate the cohorts we specified in the `cohortSetReference` described earlier. To do this, we need to communicate with the WebAPI instance, as well as the database server.

To connect to the WebAPI, we need to provide the base URL. This is a URL that looks something like "http://server.org:80/WebAPI". If you do not know the WebAPI's base URL, contact the ATLAS administrator.

We have the option to also generate inclusion rule statistics while the cohorts are instantiated (recommended). If we want to do this, we need to provide a folder where the inclusion rule statistics will be stored for later use.

To instantiate the cohorts:

```
baseUrl <- "http://server.org:80/WebAPI"
inclusionStatisticsFolder <- "c:/temp/incStats/"

instantiateCohortSet(connectionDetails = connectionDetails,
                     cdmDatabaseSchema = cdmDatabaseSchema,
                     oracleTempSchema = oracleTempSchema,
                     cohortDatabaseSchema = cohortDatabaseSchema,
                     cohortTable = cohortTable,
                     baseUrl = baseUrl,
                     cohortSetReference = cohortSetReference,
                     generateInclusionStats = TRUE,
                     inclusionStatisticsFolder = inclusionStatisticsFolder)
```

This command will contact the WebApi to obtain the cohort definitions, instantiate them in the cohort table, and write the inclusion rule statistics to the specified folder.

2.5 Generating the diagnostics

Next we generate the cohort diagnostics:

```
databaseId <- "MyData"
exportFolder <- "c:/temp/export"

runCohortDiagnostics(baseUrl = baseUrl,
                     cohortSetReference = cohortSetReference,
                     connectionDetails = connectionDetails,
                     cdmDatabaseSchema = cdmDatabaseSchema,
                     oracleTempSchema = oracleTempSchema,
                     cohortDatabaseSchema = cohortDatabaseSchema,
                     cohortTable = cohortTable,
                     inclusionStatisticsFolder = inclusionStatisticsFolder,
                     exportFolder = exportFolder,
                     databaseId = databaseId,
                     runInclusionStatistics = TRUE,
                     runIncludedSourceConcepts = TRUE,
                     runOrphanConcepts = TRUE,
                     runTimeDistributions = TRUE,
                     runBreakdownIndexEvents = TRUE,
                     runIncidenceRate = TRUE,
                     runCohortOverlap = TRUE,
                     runCohortCharacterization = TRUE,
                     minCellCount = 5)
```

The databaseId is a short string that will be used to identify the data from this database in the Shiny app. Make sure to give it a name you can easily recognize.

Once completed, a zip file will have been created in the specified export folder. This zip file can be shared between sites, as it does not contain patient-identifiable information. Note that cell counts smaller than 5 have been removed, as specified using the `minCellCount` argument, to ensure non-identifiability.

3 Viewing the diagnostics

Assuming you completed the steps described above for one or more databases, you should now have a set of zip files, one per database. Make sure to place all zip files in a single folder, for example `c:/temp/allZipFiles`.

Optionally, we can pre-merge the zip files. Merging the files can take a while, and doing it beforehand will speed up starting the Shiny app:

```
preMergeDiagnosticsFiles("C:/temp/allZipFiles")
```

To view the diagnostics, use

```
launchDiagnosticsExplorer("C:/temp/allZipFiles")
```

This will launch the Shiny app shown below:

Note that each of the tabs on the left has an information icon. Clicking on these icons will show additional information on each diagnostic, how they were computed, and how they should be interpreted.

3.1 Deploying the Shiny app on a Shiny Server

To share the results with others, it may make sense to make the Shiny app available through a Shiny Server. On the Shiny Server, a folder should be created for the app. The following should be placed in this folder:

Figure 1: The Diagnostics Explorer Shiny app

- The Shiny app itself. This can be found here. Make sure to copy all files in the `DiagnosticsExplorer` folder.
- The diagnostics data should be placed in a subfolder that should be called `data`. In this data folder, either place all the individual zip files generated for each database, or (recommended) place the pre-merged .RDA file here.

Once the app and data are in place, the Shiny Server should automatically detect them and launch the app. You may need to install additional packages required by the app, including 'shinydashboard', 'VennDiagram', 'htmltools', and 'DT'.