Using Capr

Contents

Capr provides R users with a language for building computable cohort definitions; i.e. definitions that can be translated to SQL code and executed on a observational health database in the OMOP Common Data Model format.

While some familiarity with the OMOP Common Data Model standard and the Observational Health Data Science and Informatics (OHDSI) open source ecosystem is recommended, this tutorial assumes very little prior knowledge.

We define a "cohort" as a set of persons who meet a set of inclusion criteria for a duration of time. A cohort can be thought of a set of person-time, the time during which persons in a database met the cohort definition.

A cohort is built by identifying a set of potential index dates and then applying inclusion criteria to those potential index dates. Finally persons exit a cohort either at an explicitly specified time (e.g. X days after index) or when they are no longer under observation.

In the OMOP CDM data elements are standardized so every OMOP CDM database uses the same codes to represent identical clinical data elements. Codes can be looked up in a publicly available vocabulary search tool called Athena.

Let's jump into some simple cohort definitions with Capr. We can use an example database called `Eunomia` to create reproducible code examples.

0.1 Building a concept set

The condition concept ID for Gastrointestinal hemorrhage is 192671. We can create a concept set with the `cs` function which works similar to the `c` function for creating vectors in R.

```r
library(Capr)

GIBleed <- cs(descendants(192671), name = "GIbleed")

GIBleed
```

The GIBleed concept set will include all descendants of the Gastrointestinal hemorrhage concept 192671.

0.2 Creating a cohort

Creating a simple cohort with the index date at the first occurrence of a GI bleed condition occurrence is done in a single line of R code. This cohort takes advantage of many defaults that match the defaults in Atlas.

```r
giBleedCohort <- cohort(
  entry = entry(
    conditionOccurrence(GIBleed),
    observationWindow = continuousObservation(0L, 0L),
    primaryCriteriaLimit = "First"
  ),
  exit = exit(
    endStrategy = observationExit()
  )
)

giBleedCohort
```

Cohort definitions may get more complex than this example. In general there are four slots that need to be filled for a Capr cohort: first, entry which defines the index event; second, attrition which defines the persons that are withdrawn from the cohort given inclusion or exclusion criteria; third, exit which defines when the person leaves the cohort and is no longer observation; and fourth, the era which defines the span of time of which successive periods are combined into one as well as the truncation of the patient timeline. Capr allows you to define different permutations of this cohort structure given the clinical idea under analysis for the study.

0.3 Generating a cohort

Capr is only responsible for the definition of the cohort. Capr ends when it creates a Capr Cohort object or coerces it into a json structure. At this point, if we want to generate the cohort we would switch over to different HADES software, in particular CirceR and CohortGenerator. These two R packages will help convert the json definition into executable SQL to run on your database. A generated cohort in a table will have an nx4 structure with the following columns: *cohort_definition_id, subject_id, cohort_start_date, cohort_end_date*. This table is what is used to do subsequent portions of an OHDSI study. Below is an example of how we can take the cohort from Capr and generate it on our database. For more information on generation, follow the examples from CohortGenerator

```

connectionDetails <- Eunomia::getEunomiaConnectionDetails()

giBleedCohortJson <- as.json(giBleedCohort)

sql <- CirceR::buildCohortQuery(
  expression = CirceR::cohortExpressionFromJson(giBleedCohortJson),
  options = CirceR::createGenerateOptions(generateStats = FALSE)
)

cohortsToCreate <- tibble::tibble(
  cohortId = 1,
  cohortName = "GI Bleed",
  sql = sql
)


cohortTableNames <- CohortGenerator::getCohortTableNames(cohortTable = "my_cohort_table")
CohortGenerator::createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = "main",
  cohortTableNames = cohortTableNames
)
# Generate the cohorts
cohortsGenerated <- CohortGenerator::generateCohortSet(
```

```r
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = "main",
  cohortDatabaseSchema = "main",
  cohortTableNames = cohortTableNames,
  cohortDefinitionSet = cohortsToCreate
)

# Get the cohort counts
cohortCounts <- CohortGenerator::getCohortCounts(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = "main",
  cohortTable = cohortTableNames$cohortTable
)
```