Running A Single Prevalence Analysis

Kelly Li Martin Lavallee

Contents

0.1 Overview

This tutorial walks you through running a `CohortPrevalence` for a single yearly prevalence analysis.

0.2 Connection Details

We use the `ClinicalCharacteristics` package to specify execution settings for the analysis with information on our databases and schemas of interest, and connect to the databases using `DatabaseConnector`.

```
# Create Connection details via DatabaseConnector
connectionDetails <- DatabaseConnector::createConnectionDetails(
  dbms = "dbms",
  user = "ulysses",
  password = "shh_secret"
)

# create execution Settings
executionSettings <- ClinicalCharacteristics::createExecutionSettings(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema, # schema containing patient data
  workDatabaseSchema = workDatabaseSchema, # schema to write to
  tempEmulationSchema = tempEmulationSchema, # schema to write temporary tables to
  cohortTable = cohortTable, # table on the workDatabaseSchema containing cohort data
  cdmSourceName = cdmSourceName # human-readable database source name
)

connection <- DatabaseConnector::connect(connectionDetails)
```

0.3 Defining cohorts

In this test case, we use `CapR` to generate a cohort of hypertension patients in our OMOP database.

```
library(Capr)

# make Capr concept set for Esse
```

```r
hypertensiveDisorder <- cs(
  descendants(316866),
  name = "Hypertensive disorder"
)
#fill out concept set details from vocabulary
hypertensiveDisorder <- getConceptSetDetails(
  x = hypertensiveDisorder,
  con = connection,
  vocabularyDatabaseSchema = executionSettings$cdmDatabaseSchema
)

# make Capr cohort all by all to calculate prevalence
cohort <- cohort(
  entry = entry(
    conditionOccurrence(conceptSet = hypertensiveDisorder),
    primaryCriteriaLimit = "All"
  ),
  attrition = attrition(
    expressionLimit = "All"
  ),
  exit = exit(
    endStrategy = fixedExit(index = "start", offsetDays = 0)
  )
)

# prepare cohort for CohortGenerator
json <- compile(cohort, pretty = TRUE)
sql <- CirceR::buildCohortQuery(
  expression = CirceR::cohortExpressionFromJson(json),
  options = CirceR::createGenerateOptions(generateStats = FALSE)
)

cohortDefinitionSet <- data.frame(
  cohortId = 316866,
  cohortName = "hypertension",
  json = json,
  sql = sql
)

# build cohort tables for generation
cohortTableNames <- CohortGenerator::getCohortTableNames(cohortTable = executionSettings$cohortTable)

CohortGenerator::createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = executionSettings$workDatabaseSchema,
  cohortTableNames = cohortTableNames
)

# generate cohorts
CohortGenerator::generateCohortSet(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = executionSettings$workDatabaseSchema,
  tempEmulationSchema = executionSettings$tempEmulationSchema,
```

```
  cohortTableNames = cohortTableNames,
  cdmDatabaseSchema = executionSettings$cdmDatabaseSchema,
  cohortDefinitionSet = cohortDefinitionSet
)

# Check counts
cohortCounts <- CohortGenerator::getCohortCounts(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = executionSettings$workDatabaseSchema,
  cohortTable = cohortTableNames$cohortTable
)

cohortCounts
```

0.4 Step 1: Cohorts and periods of interest

The first step of the analysis is to specify the cohort of prevalent interest and the yearly periods of interest. In order to do so, we create the `R6` classes that define these analyses settings. Sometimes, we may want to do an analysis on a subpopulation of the overall database (i.e. prevalence of hypertension in sitagliptin users). In this analysis, we care about the entire population, and so we leave the `populationCohort` option `NULL`.

```
prevalentCohort <- createPrevalenceCohort(cohortId = 316866,
                                          cohortName = "Hypertension")
periodOfInterest <- createYearlyPrevalence(range = c(2016:2020))
populationCohort <- NULL
```

0.5 Step 2: Prevalence analysis options

Next, we define the options specific to the prevalence analysis. This includes a choice of numerator and denominator computational technique. `CohortPrevalence` uses operational definitions of prevalence from Rassen et al. Please see the vignette `prevalence` for definitions of the numerator and denominator choices.

Beyond the numerator and denominator, we also create options for the length of lookback (and whether we only want to use observed time in the lookback), rate multiplier, minimum observation length, strata variables (default age and gender), and whether or not we want to use only the first observation period,

```
analysisId <- 123 # Any unique integer ID to define this analysis

# Select numerator and denominator options
numeratorType <- "pn1" #pn1 or pn2
denominatorType <- createDenominatorType(denomType = "pd3") #pd1, pd2, pd3, or pd4

# Set lookback period options
lookBackOptions <- createLookBackOptions(lookBackDays = 99999L,
                                         useObservedTimeOnly = FALSE)

# Set strata options - NULL by default uses age and gender
strata <- NULL

# Set other specifications
minimumObservationLength <- 0L
useOnlyFirstObservationPeriod <- FALSE
multiplier <- 100000

prevalenceAnalysisClass <- createCohortPrevalenceAnalysis(
```

```
  analysisId = analysisId,
  prevalentCohort = prevalentCohort,
  periodOfInterest = periodOfInterest,
  lookBackOptions = lookBackOptions,
  numeratorType = numeratorType,
  denominatorType = denominatorType,
  minimumObservationLength =  minimumObservationLength,
  useOnlyFirstObservationPeriod =  useOnlyFirstObservationPeriod,
  multiplier = multiplier,
  strata = strata,
  populationCohort = NULL
)
```

0.6 Step 3: Run analyses

The `prevalenceAnalysisClass` object is now a wrapped-up package of all our analysis specifications. Using this object, we can now run the analysis with `generateSinglePrevalence` and our previously defined database connection settings.

```
# Results
results <- generateSinglePrevalence(
  prevalenceAnalysisClass = prevalenceAnalysisClass,
  executionSettings = executionSettings
)
```

0.7 Step 4: Export

Now, we can write the results dataframe into `.csv` format for sharing results with `exportPrevalenceResults` Sometimes, we want to externally review the Sql queries used to generate the prevalence objects. We can do this with `exportPrevalenceQuery`.

```
outputFolder <- here::here("results") |>
  fs::dir_create()

# Export results to CSV format
exportPrevalenceResults(
  results = results,
  outputFolder = outputFolder
)

# Save SQL query
exportPrevalenceQuery(
  prevalenceAnalysisClass = prevalenceAnalysisClass,
  outputFolder = outputFolder
)
```