Sampling Cohorts

James P. Gilbert

2025-09-05

Contents

Sampling with CohortGenerator

Large populations of individuals (e.g. all subjects receiving a COVID-19 vaccination) can often be too large to work with when pulling down a large collection of covariates for further analysis. This is prohibitive when designing studies or attempting to generate phenotypes. This guide aims to demonstrate how one can use the `sampleCohortDefinitionSet` functionality to produce sufficiently large sample cohorts from a base `cohortDefinitionSet`.

Sampling method

The approach taken in this method is to sample individuals within a cohort without replacement. Different database platforms implement a variety of different approaches for random sampling and random number generation that make cross platform sql difficult. Consequently, all the randomness computed here happens inside `R` itself simply from the count of unique individuals within a cohort.

Using the sampler functions

To create a single sample the approach below will create cohorts in the same base table.

First we need to load the initial cohort definition set

```
cds <- getCohortDefinitionSet(...)
```

We then need to create the cohort tables and cohorts in the usual manner.

```
connectionDetails <- Eunomia::getEunomiaConnectionDetails()
conn <- DatabaseConnector::connect(connectionDetails = connectionDetails)
on.exit(DatabaseConnector::disconnect(conn))



cds <- getCohortDefinitionSet(...)
cohortTableNames <- getCohortTableNames(cohortTable = "cohort")
recordKeepingFolder <- file.path(outputFolder, "RecordKeepingSamples")

createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = "main",
```

```
  cohortTableNames = cohortTableNames
)

generateCohortSet(
  cohortDefinitionSet = cds,
  connection = conn,
  cdmDatabaseSchema = "main",
  cohortDatabaseSchema = "main",
  cohortTableNames = cohortTableNames,
  incremental = TRUE,
  incrementalFolder = recordKeepingFolder
)
```

We can then create a new cohort definition set from the original sample.

```
sampledCohortDefinitionSet <- sampleCohortDefinitionSet(
  cohortDefinitionSet = cds,
  connection = conn,
  sampleFraction = 0.33,
  seed = 64374, # OHDSI
  cohortDatabaseSchema = "main",
  cohortTableNames = cohortTableNames,
  incremental = TRUE,
  incrementalFolder = recordKeepingFolder
)
```

The resulting `sampledCohortDefinitionSet` is nearly identical to the base cohort set, however a few changes occur:

- The name now include the postfix `[sample 33% seed=64374]`
- The optional parameter `idExpression` changes the cohort name. By default this is set to `cohortId * 1000 + seed` however, this will throw an error if the ids are the same
- The
- The base cohortDefinitionSet is attached as an attribute to the `sampledCohortDefinitionSet`

To generate multiple samples, simply specify multiple seed variables as follows:

```
# Generate 800 samples of size n
sampledCohortDefinitionSet <- sampleCohortDefinitionSet(
  cohortDefinitionSet = cds,
  connection = conn,
  n = 1000,
  seed = 1:800 * 64374, # OHDSI
  cohortDatabaseSchema = "main",
  cohortTableNames = cohortTableNames,
  incremental = TRUE,
  incrementalFolder = recordKeepingFolder
)
```

Note that using incremental mode for your sampled cohorts will also work. In this case, a cohort will only be re-generated if the checksum of the base cohort has changed (the checksum is based on the cohort SQL). The checksum applies to the pseudorandom seed of the cohort and the sample size (n).

As the sampledCohortDefinitionSet is, for all intents and purposes, a set of cohortDefinitions it can be passed as a parameter to all other OHDSI packages with minimal issues. For example, `FeatureExtraction` will be able to use this sample unchanged.

```
options(old)
```