How to use ClinicalCharacteristics

Martin Lavallee Katy Sadowski Ajit Londhe Ron Herrera

Contents

1 Introduction

When conducting real-world evidence (RWE) studies, analysts often need to populate table-shells defined by clinical colleagues. In principle, these table-shells should be simple to generate, as they consist primarily of counts and percentages for categorical variables and summary statistics for continuous variables. Surprisingly, this task proves to be non-trivial when using the available OHDSI tools. The best available tool for characterization is `FeatureExtraction` which is a tool designed to extract as many clinical features as possible to use as covariates for high-dimensional modelling. While helpful for modelling, populating table-shells is awkward because it pulls all concepts from a domain and performs characterization at the individual concept level. One could also use the web application ATLAS to characterize, however, it does not fit programmatic driven pipelines. With some of these gaps in mind, we designed a new OHDSI tool titled `ClinicalCharacteristics` which is a table shell approach to OMOP characterization.

2 Features

A characterization requires the identification of a clinical event during a specified window of time relative to the index date of the cohort of interest. For example if we wish to look at baseline characteristics, we anchor our enumeration on the date at which the person enters the cohort and consider a window of time as baseline (say -365 days to -1 days relative to index). There are a few assumptions we need to make from this construction: a) what type of event are we looking for, b) when are we looking, relative to index, and must it be during observed time and c) how do we want to summarize this information.

2.1 Data Types

`ClinicalCharacteristics` can summarize the following types of OHDSI objects:

- **Demographics**: information rooted in the person table such as age, gender, race, and ethnicity.
- **Concept Sets**: groupings of concepts in a single domain, applying hierarchical logic to identify large lists of clinical concepts. For example a Type 2 Diabetes concept set would include the high-level concept and its descendants.
- **Concept Set Groups**: groups of concepts across multiple domains. Sometimes concepts for an occurrence do not occur in one domain (i.e conditions, drugs, procedures) but in multiple. For example, smoking status occurs in observation, procedure, and condition tables. This item unifies them as a single entity.
- **Cohorts**: identification of clinical event across a set of persons over an era of time bracketed by an entry and exit event. Cohorts apply the occurrence of an event accounting for attrition based on inclusion/exclusion criteria.
- **Source Codes**: [Future Addition] a list of non-standard codes under a single vocabulary to identify events. It is not recommended to use this type of object because it does not leverage the advantages of semantic standardization
- **Other**: [Future Addition] Sometimes clinically relevant events don't fall into the these categories and require special constructions to query in the CDM. Items such as lab measurements, locations and episodes.

2.2 Value Types

Beyond types of domains to summarize, there are different ways of performing an aggregation of information. `ClinicalCharacteristics` can create tables based on these value types:

- **Presence**: an event is either observed or not observed for a patient given a time window. With this statistic type we report a categorical value as n and percent.
- **Continuous Distribution**: how many times is an event or measured value observed during a period of time. With this statistic type we report as a continuous value using mean, standard deviation and default order statistics
- **Breaks**: sometimes we want to convert a continuous value into a categorical grouping. For example persons taking 1, 2, 3, or 4+ medications during a period of time. We apply breaks to categorize counts and report as a categorical value with n and percent
- **Score**: [Future Addition] the inverse of breaks, we may want to turn a categorical grouping into a continuous value. For example we wish to apply a weight to the presence of an event and the summarize a collection of scores per patient.

2.3 Observed Time

Another consideration is whether to summarize occurrence based on any time or observed time. When we define a time window for characterization this does not consider whether the persons were under continuous observation. To add this feature in we need to specify to count only persons who were actively followed during the interval in both the numerator and denominator. `ClinicalCharacteristics` specifies if we enumerate at **any** time (naively apply the time window) or **observed** (apply the time window in the context of observed time). It is recommended to use **any** time for most characterization, as it is the simplest to interpret.

3 Setup

3.1 Execution Settings

As do all OHDSI tools we need to setup the database credentials of the dbms hosting the OMOP data we wish to use with `ClinicalCharacteristics`. We use the HADES package `DatabaseConnector` to create a `connectionDetails` object that can be used to establish a database connection. Below we provide an example of setting up connection details, where users can find more references from the `DatabaseConnector` documentation.

```
# set a environment variable defining the jar folder path
Sys.setenv("DATABASECONNECTOR_JAR_FOLDER" = "c:/temp/jdbcDrivers")

# download the jdbc driver
DatabaseConnector::downloadJdbcDrivers("postgresql")

connectionDetails <- DatabaseConnector::createConnectionDetails(
  dbms = "postgresql",
  user = "joe",
  password = "secret",
  connectionString = "jdbc:postgresql://localhost:5432/postgres"
)

# test the connection
con <- DatabaseConnector::connect(connectionDetails)
DatabaseConnector::disconnect(con)
```

Something we are introducing within the `ClinicalCharacteristics` package is a class called `ExecutionSettings`. This object maintains all the settings needed to execute a query on your dbms. Key parameters include:

- **cdmDatabaseSchema**: a read only schema where the OMOP CDM is located
- **workDatabaseSchema**: a read/write schema where you will conduct work; effectively the location of the cohort table
- **tempEmulationSchema**: a read/write schema to store temp tables. Note this is only necessary for dbms' like snowflake and oracle that handle temp tables differently than other dialects
- **targetCohortTable**: the name of the cohort table where your target cohorts exist
- **cdmSourceName**: a label for the CDM in use

In `ClinicalCharacteristics` we can set up our execution settings object as follows:

```
executionSettings <- createExecutionSettings(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = "omop_schema",
  workDatabaseSchema = "my_schema",
  tempEmulationSchema = "my_schema",
  targetCohortTable = "my_cohorts",
  cdmSourceName = "my_omop"
)
```

The convenience of this object is that it serves as a container to pass parameters to steps that execute the sql. Note users also do not need to worry about opening and closing connections when using these tools.

Replacing parameters in the executionSettings object is done through simple assignment:

```
executionSettings$cdmSourceName <- "my_other_omop"
executionSettings$workDatabaseSchema <- "my_other_schema"
```

3.2 Cohorts

`ClinicalCharacteristics` works under the assumption that cohorts have already been instantiated in the cohort table. For new OHDSI users, please review the `CohortGenerator` package on how to instantiate cohorts in the cohort table.

3.3 Concept Sets

Our key distinguishing feature to the `FeatureExtraction` package is the ability to characterize using Concept Sets as opposed to individual concept ids. There are two ways to import concept sets into the `ClinicalCharacteristics` work flow: a) import from json or b) create on the fly. Both methods apply the R package `Capr`.

3.3.1 Import from Json

Users can export a concept set definition json from ATLAS and save as a file with json extension. To import the concept set into R as an object do as follows:

```
conceptSetFile <- here::here("my_concept_set.json")
cs1 <- Capr::readConceptSet(
  path = conceptSetFile,
  name = "My Concept Set"
)
```

We recommend using this method if you have complex concept sets.

3.3.2 Create on the Fly

Also using `Capr` one can create a simple concept set using the standard concept id of interest.

```
cs2 <- Capr::cs(Capr::descendants(201826), name = "t2d")
```

This function creates a concept set class in Capr. We take the standard concept for type 2 diabetes and include its descendants to make a concept set. We recommend this method when building simple concept sets.

4 Using ClinicalCharacteristics

Now that we have setup the requirements for `ClinicalCharacteristics` we can begin to use the program to develop table shells. The goal is to populate a `TableShell` class object with the details of how you want to populate your table. The key elements of the table shell are: the title, the target cohorts to use and the table line items.

4.1 Target Cohorts

First we must specify which cohorts we want to characterize in our shell. Assuming they have been instantiated in a specific cohort table, we can load them as follows:

```
targetCohorts <- list(
    createCohortInfo(id = 1, name = "T2D"),
    createCohortInfo(id = 2, name = "T1D")
  )
```

We list all the cohorts we want to use. You may add as many instantiated cohorts as you would like into this list and `ClinicalCharacteristics` will render the same table shell per target cohort.

4.2 Line Items

Each element of the table is called a line item. The line item uniquely summarizes events with a particular statistic type at a specified time window. For example, if we want to characterize CKD we need to specify the line item:

```
ckd_cs <- Capr::cs(Capr::descendants(46271022), name = "ckd")
createConceptSetLineItem(
  sectionLabel = "CKD: Baseline",
  domain = "condition_occurrence",
  statistic = anyPresenceStat(),
  conceptSet = ckd_cs,
  timeInterval = tw1
)
```

The section label specifies how we want to identify the line item within the table as a section. The domain must specify which domain table in the CDM we should check for events. Because CKD is a condition, we want to check the *condition_occurrence* table. If we were characterizing drugs we would check the *drug_exposure* table. Review the OMOP CDM to become familiar with the different types of domains. The statistic determines how to summarize the event; here are examples:

- `PresenceStat`: determine who had an occurrence of an event during an interval
- `CountCtsStat`: determine the continuous distribution of an event during an interval
- `CountBreaksStat`: convert the count to breaks of an event during an interval
- `TimeToFirstCtsStat`: determine the continuous distribution of the time to the occurrence of the first event
- `TimeToFirstBreaksStat`: convert the time to the first event into a break of values

To distinguish between **any** and **observed** add the keyword as a prefix (i.e. `anyPresenceStat()` or `observedPresenceStat()`).

4.3 Basic Example

Let's use the following details for an example table shell, assuming a cohort for type 2 diabetes has been created:

- male gender
- age as continuous
- presence of condition ckd -365d to -1d prior to index
- presence of condition t2d -365d to -1d prior to index
- presence of drug sglt2 0d to 90d post index

```
library(ClinicalCharacteristics)

# Define Execution Settings
executionSettings <- createExecutionSettings(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = "omop_schema",
  workDatabaseSchema = "my_schema",
  tempEmulationSchema = "my_schema",
  targetCohortTable = "my_cohorts",
  cdmSourceName = "my_omop"
)

# create each concept set
ckd_cs <- Capr::cs(Capr::descendants(46271022), name = "ckd")
hf_cs <- Capr::cs(Capr::descendants(316139), name = "hf")
```

```r
sglt2_cs <- Capr::cs(Capr::descendants(1123627), name = "sglt2")

# make each time window
tw1 <- timeInterval(lb = -365, rb = -1)
tw2 <- timeInterval(lb = 0, rb = 90)

# define table shell
ts <- createTableShell(
  title = "Demo 1",
  targetCohorts = list(
    createCohortInfo(id = 1, name = "Type 2 Diabetes")
  ),
  lineItems = lineItems(
    createDemographicLineItem(maleGender()),
    createDemographicLineItem(ageChar()),
    createConceptSetLineItem(
      sectionLabel = "CKD: Baseline",
      domain = "condition_occurrence",
      statistic = anyPresenceStat(),
      conceptSet = ckd_cs,
      timeInterval = tw1
    ),
    createConceptSetLineItem(
      sectionLabel = "HF: Baseline",
      domain = "condition_occurrence",
      statistic = anyPresenceStat(),
      conceptSet = hf_cs,
      timeInterval = tw1
    ),
    createConceptSetLineItem(
      sectionLabel = "SGLT2: Post Index",
      domain = "drug_exposure",
      statistic = anyPresenceStat(),
      conceptSet = sglt2_cs,
      timeInterval = tw2
    )
  )
)

res <- generateTableShell(ts, executionSettings)
res$categorical
res$continuous
```

4.4 Conveniences

Line items are an important feature of `ClinicalCharacteristics` in order to track the table shell in a consistent matter. However, from a UI perspective this becomes very "typing" intensive. So we have added certain convenience functions to make it easier to design the table shell. Taking the same example from above, what if we added the following to our table shell:

- add female gender
- race categories
- age in 5-year breaks
- sglt2 and glp1 at 0-90 and 0-365 days post index

```r
library(ClinicalCharacteristics)

# Define Execution Settings
executionSettings <- createExecutionSettings(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = "omop_schema",
  workDatabaseSchema = "my_schema",
  tempEmulationSchema = "my_schema",
  targetCohortTable = "my_cohorts",
  cdmSourceName = "my_omop"
)

# create each concept set
condition_cs <- list(
  Capr::cs(Capr::descendants(46271022), name = "ckd"),
  Capr::cs(Capr::descendants(316139), name = "hf")
)

drug_cs <- list(
  Capr::cs(Capr::descendants(1123627), name = "sglt2")
  Capr::cs(Capr::descendants(1123618), name = "sglt2")
)

# make each time window
tw1 <- list(
  timeInterval(lb = -365, rb = -1)
)
tw2 <- list(
  timeInterval(lb = 0, rb = 90),
  timeInterval(lb = 0, rb = 365)
)

# define table shell
ts2 <- createTableShell(
  title = "Demo 1",
  targetCohorts = list(
    createCohortInfo(id = 1, name = "Type 2 Diabetes")
  ),
  lineItems = lineItems(
    addDefaultGenderLineItems(),
    addDefaultEthnicityLineItems(),
    createDemographicLineItem(ageChar()),
    createDemographicLineItem(ageChar(breaks = age5yrGrp())),
    createConceptSetLineItemBatch(
      sectionLabel = "Baseline Conditions",
      domain = "condition_occurrence",
      statistic = anyPresenceStat(),
      conceptSets = condition_cs,
      timeIntervals = tw1
    ),
    createConceptSetLineItemBatch(
      sectionLabel = "Post-Index Drugs",
      domain = "drug_exposure",
```

```
      statistic = anyPresenceStat(),
      conceptSets = drug_cs,
      timeIntervals = tw2
    )
  )
)

res <- generateTableShell(ts2, executionSettings)
res$categorical
res$continuous
```

In this version of the table shell we have grouped drugs and conditions as a batch to deploy all combinations of the concept set and time interval combinations. As for the demographics we add a convenience function to include all basic gender items (male and female) and race items (white, black, asian and not reported). Finally we apply our first instance of a breaks strategy, turning age into 5 year age groups. A convenience function is used that automatically sets the breaks for the user.

Note when a user generates the table shell a log of the shell is printed out for review. Users may also review the table shell meta information to understand the aspects of each line item using the following method of the table shell class which returns a tibble of information.

When using **multiple concepts and/or multiple time windows**, the Batch functions must be used otherwise an error will display.

```
ts2$getTableShellMeta()
```

4.5 Cohorts to Describe

Sometimes we are interested in cohorts as items to characterize in a table shell. For example, say we wanted to describe persons with heart failure who also have ckd (at baseline). This is more easily done as a cohort rather than a concept set beacuse we have to apply logic to the population.

In order to use cohorts in characterization, users must generate the cohort in a cohort table in their workDatabaseSchema prior to usage. Assuming that is done, adding this feature into the table shell is quite simple.

```
library(ClinicalCharacteristics)

# Define Execution Settings
executionSettings <- createExecutionSettings(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = "omop_schema",
  workDatabaseSchema = "my_schema",
  tempEmulationSchema = "my_schema",
  targetCohortTable = "my_cohorts",
  cdmSourceName = "my_omop"
)

# create each concept set
condition_cs <- list(
  Capr::cs(Capr::descendants(46271022), name = "ckd"),
  Capr::cs(Capr::descendants(316139), name = "hf")
)

drug_cs <- list(
  Capr::cs(Capr::descendants(1123627), name = "sglt2")
```

```
  Capr::cs(Capr::descendants(1123618), name = "glp1")
)

covCohorts <- list(
  createCohortInfo(id = 3, name = "HF with CKD at baseline"),
  createCohortInfo(id = 4, name = "Obesity with CKD at baseline")
)

# make each time window
tw1 <- list(
  timeInterval(lb = -365, rb = -1)
)
tw2 <- list(
  timeInterval(lb = 0, rb = 90),
  timeInterval(lb = 0, rb = 365)
)

# define table shell
ts3 <- createTableShell(
  title = "Demo 1",
  targetCohorts = list(
    createCohortInfo(id = 1, name = "Type 2 Diabetes")
  ),
  lineItems = lineItems(
    addDefaultGenderLineItems(),
    addDefaultRaceLineItems(),
    createDemographicLineItem(ageChar()),
    createDemographicLineItem(ageChar(breaks = age5yrGrp())),
    createConceptSetLineItemBatch(
      sectionLabel = "Baseline Conditions",
      domain = "condition_occurrence",
      statistic = anyPresenceStat(),
      conceptSets = condition_cs,
      timeIntervals = tw1
    ),
    createConceptSetLineItemBatch(
      sectionLabel = "Post-Index Drugs",
      domain = "drug_exposure",
      statistic = anyPresenceStat(),
      conceptSets = drug_cs,
      timeIntervals = tw2
    ),
    createCohortLineItemBatch(
      sectionLabel = "Baseline Cohorts",
      covariateCohorts = covCohorts,
      cohortTable = "my_cohort_table",
      timeIntervals = tw1,
      statistic = anyPresenceStat()
    )
  )
)

res <- generateTableShell(ts3, executionSettings)
```

```
res$categorical
res$continuous
```

Notice the UI for adding cohorts into the table shell as line items is the same as the target cohorts. We make CohortInfo class objects as a list. In the lineItem or lineItemBatch calls we must specify into which table these cohorts are instantiated.

4.6 Concept Set Groups

Sometimes clinical events can be described in multiple domains. Smoking for example has concepts in the observation, procedure and condition occurrence tables. Handling multiple domains is easiest done as a group, where we specify each domain to look for our concept set. The code below creates a concept set group line item that can be added to a table shell.

```
smoking_cs <- list(
  Capr::cs(Capr::descendants(761009), name = "Smoking Assessment"),
  Capr::cs(Capr::descendants(44804450), name = "Smoking Cessation")
)

csg <- createConceptSetGroupLineItem(
  groupLabel = "Smoking",
  conceptSets = smoking_cs,
  domainTables = c("procedure_occurrence", "observation"),
  statistic = anyPresenceStat(),
  timeInterval = timeInterval(lb = -365, rb = -1)
)
```

Notice we specify the domain for each element of the concept set list. When enumerating this will collapse into one category instead of two items.

4.7 Breaks

It is very common to want to convert a continuous summary into a categorical summary. Say we want to know how many persons took 1, 2, 3, or 4+ of a medication like sglt2 or glp1s, instead of how many took at least 1 of these drugs at any time. To do this we must make a breaks strategy.

```
drug_cs <- list(
  Capr::cs(Capr::descendants(1123627), name = "sglt2")
  Capr::cs(Capr::descendants(1123618), name = "glp1")
)

tw2 <- list(
  timeInterval(lb = 0, rb = 90),
  timeInterval(lb = 0, rb = 365)
)

br <- newBreaks(
  name = "Medication Groups",
  breaks = c(1,2,3,4),
  labels = c("one", "two", "three", "four or more")
)

drugBreaks <- createConceptSetLineItemBatch(
  sectionLabel = "Drug CountBreaks",
  domain = "drug_exposure",
```

```
  statistic = anyCountBreaksStat(breaks = br),
  conceptSets = drug_cs,
  timeIntervals = tw2
)
```

Once you have built this line item batch it can be added into the table shell. Instead of providing a continuous summary, these will now be broken into categories as specified in the `newBreaks` function. If we actually wanted the continuous summary the code would look as follows:

```
drug_cs <- list(
  Capr::cs(Capr::descendants(1123627), name = "sglt2")
  Capr::cs(Capr::descendants(1123618), name = "glp1")
)

tw2 <- list(
  timeInterval(lb = 0, rb = 90),
  timeInterval(lb = 0, rb = 365)
)

drugBreaks <- createConceptSetLineItemBatch(
  sectionLabel = "Drug CountBreaks",
  domain = "drug_exposure",
  statistic = anyCountCtsStat(),
  conceptSets = drug_cs,
  timeIntervals = tw2
)
```

4.8 Review the Sql

The way `ClinicalCharacteristics` works is it serializes the input of the TableShell object into a Sql script that is run against the database connection. Sometimes users may want to review the sql to either check its quality or modify it slightly. We provide a function to save the script designed by the program as a `.sql` that can be opened in any code or text editor.

```
reviewTableShellSql(ts, executionSettings, saveName = "my_ts", savePath = here::here("output"))
```

4.9 Using source concepts

In certain circumstances, researchers might need to use source concepts instead of standardized codes, however, the recommendation is always to use standardized codes.

4.10 Searching the codes

On occasion we can find the OHDSI concept id using the function `lookupSourceConcepts`. For instance, we would like to use the ICD10CM codes: 'I45.81', 'I47', 'I49', 'I49.40 to build a *Cardiac Arrhythmia* concept set as follows:

```
ca_source <- lookupSourceConcepts(
  codes = c('I45.81', 'I47', 'I49', 'I49.40'),
  vocabulary = "ICD10CM",
  executionSettings
) |>
  sourceConceptSet(name = "Cardiac Arrhythmia")
```

Then the table shell will look like

```r
tw1 <- timeInterval(lb = -365, rb = -1)

tssc <- createTableShell(
  title = "Demo 1",
  targetCohorts = list(
    createCohortInfo(id = 1, name = "Type 2 Diabetes")
  ),
  lineItems = lineItems(
    addDefaultGenderLineItems(),
    addDefaultRaceLineItems(),
    createDemographicLineItem(ageChar()),
    createDemographicLineItem(ageChar(breaks = age5yrGrp())),
    createSourceConceptSetLineItem(
      sectionLabel = "Source Concept Conditions Cardiac Arrhythmia",
      domain = "condition_occurrence",
      sourceConceptSets = ca_source,
      timeIntervals = tw1,
      statistic = anyPresenceStat()
    )
  )
)
```

4.11 Reading source concept from Json files

It is also possible to build the source concept set in ATLAS in a Json format. Then, we can read the file using Capr's functions as follows:

```r
ca_source <- Capr::readConceptSet(path = paste0(inputFolder, "Cardiac Arrhythmia_in_icd.json"), name =

ca_source <- Capr::as.data.frame(ca_source) |>
  dplyr::select(conceptId, conceptName, conceptCode, vocabularyId) |>
  sourceConceptSet(name = "Cardiac Arrhythmia")
```