Creating custom covariate builders

Martijn J. Schuemie

2023-11-20

Contents

1 Introduction

This vignette assumes you are already familiar with the `FeatureExtraction` package.

The `FeatureExtraction` package can generate a default set of covariates, such as one covariate for each condition found in the `condition_occurrence` table. However, for some reasons one might need other covariates than those included in the default set. Sometimes it might make sense to request the new covariates be added to the standard list, but other times there is good reason to keep them separated.

The `FeatureExtraction` package has a mechanism for including custom covariate builders to either replace or complement the covariate builders included in the package. This vignette describes that mechanism.

Note: another way to add custom covariates is by using the `cohort_attribute` table in the common data model. This approach is described in the vignette called `creating covariates using cohort attributes`, and might be more suitable if you are likely to need the covariates only once, or when you are less familiar with advanced R programming. Creating a custom covariate builder as described in this vignette is more complicated, but once completed can easily be reused in many studies.

2 Overview

To add a custom covariate builder, two things need to be implemented:

1. A function that creates a `covariateSettings` object for the custom covariates.
2. A function that uses the covariate settings to construct the new covariates.

3 Covariate settings function

The covariate settings function must create an object that meets two requirements:

1. The object must be of class `covariateSettings`.
2. The object must have an attribute `fun` that specifies the name of the function for generating the covariates.

3.1 Example function

Here is an example covariate settings function:

```
createLooCovariateSettings <- function(useLengthOfObs = TRUE) {
  covariateSettings <- list(useLengthOfObs = useLengthOfObs)
  attr(covariateSettings, "fun") <- "getDbLooCovariateData"
  class(covariateSettings) <- "covariateSettings"
  return(covariateSettings)
}
```

In this example the function has only one argument: `useLengthOfObs`. This argument is stored in the `covariateSettings` object. We specify that the name of the function that will construct the covariates corresponding to these options is `getDbLooCovariateData`.

4 Covariate construction function

4.1 Function inputs

The covariate construction function has to accept the following arguments:

- `connection`: A connection to the server containing the schema as created using the `connect` function in the `DatabaseConnector` package.
- `oracleTempSchema`: A schema where temp tables can be created in Oracle.
- `cdmDatabaseSchema`: The name of the database schema that contains the OMOP CDM instance. On SQL Server, this will specify both the database and the schema, so for example 'cdm_instance.dbo'.
- `cdmVersion`: Defines the OMOP CDM version used: currently supports "4" and "5".
- `cohortTable`: Name of the table holding the cohort for which we want to construct covariates. This is a fully specified name, so either the name of a temp table (e.g. '#cohort_table'), or a permanent table including its database schema (e.g. 'cdm_schema.dbo.cohort').
- `cohortId`: The cohort definition ID of the cohort. If set to -1, use all entries in the cohort table.
- `rowIdField`: The name of the field in the cohort temp table that is to be used as the row_id field in the output table. This can be especially usefull if there is more than one period per person.

- `covariateSettings`: The object created in your covariate settings function.
- `aggregated`: Should covariates be constructed per-person, or aggregated across the cohort?

The function can expect that a table exists with the name specified in the `cohortTable` argument. This table will identify the persons and the index dates for which we want to construct the covariates, and will have the following fields: `subject_id`, `cohort_start_date`, and `cohort_definition_id`. Because sometimes there can be more than one index date (i.e. `cohort_start_date`) per person, an additional field can be included with a unique identifier for each `subject_id` - `cohort_start_date` combination. The name of this field will be specified in the `rowIdField` argument

4.2 Function outputs

The function must return an object of type `CovariateData`, which is an Andromeda object with the following members:

- `covariates`, listing the covariates per row ID. This is done using a sparse representation; covariates with a value of 0 are omitted to save space. The covariates object must have three columns: `rowId`, `covariateId`, and `covariateValue`.
- `covariateRef`, describing the covariates that have been extracted. This should have the following columns: `covariateId`, `covariateName`, `analysisId`, `conceptId`.
- `analysisRef`, dscribing the analyses performed by the function. This should have the following columns: `analysisId`, `analysisName`, `domainIdsta`, `startDay`, `endDay`, `isBinary`, `missingMeansZero`.

Additionally, the object should have an attribute called `metaData`, which is a (potentially empty) list of objects with information on how the `CovariateData` object was constructed.

4.3 Example function

```
getDbLooCovariateData <- function(connection,
                                  oracleTempSchema = NULL,
                                  cdmDatabaseSchema,
                                  cdmVersion = "5",
                                  cohortTable = "#cohort_person",
                                  cohortId = -1,
                                  rowIdField = "subject_id",
                                  covariateSettings,
                                  aggregated = FALSE) {
  writeLines("Constructing length of observation covariates")
  if (covariateSettings$useLengthOfObs == FALSE) {
    return(NULL)
  }
  if (aggregated) {
    stop("Aggregation not supported")
  }

  # Some SQL to construct the covariate:
  sql <- paste(
    "SELECT @row_id_field AS row_id, 1 AS covariate_id,",
    "DATEDIFF(DAY, observation_period_start_date, cohort_start_date)",
```

```
    "AS covariate_value",
    "FROM @cohort_table c",
    "INNER JOIN @cdm_database_schema.observation_period op",
    "ON op.person_id = c.subject_id",
    "WHERE cohort_start_date >= observation_period_start_date",
    "AND cohort_start_date <= observation_period_end_date",
    "{@cohort_id != -1} ? {AND cohort_definition_id = @cohort_id}"
  )
  sql <- SqlRender::render(sql,
    cohort_table = cohortTable,
    cohort_id = cohortId,
    row_id_field = rowIdField,
    cdm_database_schema = cdmDatabaseSchema
  )
  sql <- SqlRender::translate(sql, targetDialect = attr(connection, "dbms"))

  # Retrieve the covariate:
  covariates <- DatabaseConnector::querySql(connection, sql, snakeCaseToCamelCase = TRUE)

  # Construct covariate reference:
  covariateRef <- data.frame(
    covariateId = 1,
    covariateName = "Length of observation",
    analysisId = 1,
    conceptId = 0
  )

  # Construct analysis reference:
  analysisRef <- data.frame(
    analysisId = 1,
    analysisName = "Length of observation",
    domainId = "Demographics",
    startDay = 0,
    endDay = 0,
    isBinary = "N",
    missingMeansZero = "Y"
  )

  # Construct analysis reference:
  metaData <- list(sql = sql, call = match.call())
  result <- Andromeda::andromeda(
    covariates = covariates,
    covariateRef = covariateRef,
    analysisRef = analysisRef
  )
  attr(result, "metaData") <- metaData
  class(result) <- "CovariateData"
  return(result)
}
```

In this example function, we construct a single covariate called 'Length of observation', which is the number of days between the `observation_period_start_date` and the index date. We use parameterized SQL and the `SqlRender` package to generate the appropriate SQL statement for the database to which we are connected. We also create the covariate reference and analysis reference objects, which have one row each, specifying our

one covariate and one analysis. We then wrap up the `covariate`, `covariateRef`, and `analysisRef` objects in a single result Andromeda object, together with some meta-data.

5 Using the custom covariate builder

We can use our custom covariate builder in the `PatientLevelPrediction` package, as well other packages that depend on the `FeatureExtraction` package, such as the `CohortMethod` package. If we want to use only our custom defined covariate builder, we can simply replace the existing `covariateSettings` with our own, for example:

```
looCovSet <- createLooCovariateSettings(useLengthOfObs = TRUE)

covariates <- getDbCovariateData(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  cohortDatabaseSchema = resultsDatabaseSchema,
  cohortTable = "rehospitalization",
  cohortId = 1,
  covariateSettings = looCovSet
)
```

In this case we will have only one covariate for our predictive model, the length of observation. In most cases, we will want our custom covariates in addition to the default covariates. We can do this by creating a list of covariate settings:

```
covariateSettings <- createCovariateSettings(
  useDemographicsGender = TRUE,
  useDemographicsAgeGroup = TRUE,
  useDemographicsRace = TRUE,
  useDemographicsEthnicity = TRUE,
  useDemographicsIndexYear = TRUE,
  useDemographicsIndexMonth = TRUE
)

looCovSet <- createLooCovariateSettings(useLengthOfObs = TRUE)

covariateSettingsList <- list(covariateSettings, looCovSet)

covariates <- getDbCovariateData(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  cohortDatabaseSchema = resultsDatabaseSchema,
  cohortTable = "rehospitalization",
  cohortId = 1,
  covariateSettings = covariateSettingsList
)
```

In this example both demographic covariates and our length of observation covariate will be generated and can be used in our predictive model.