An introduction to the AssociationRuleMining package for advanced users.

Contents

1 Overview **1**

2 Introduction **2**

3 Implementation **2**

3.1 Data Preparation 2

4 Use in other packages **5**

1 Overview

To quickly summarise how to implement the AssociationRuleMining package:

1) The package implements two analysis frameworks, association rule and frequent pattern mining.

Assuming the user has instantiated an appropriate and relevant cohort, and extracted the relevant information using `FeatureExtraction`.

2) Each analysis framework of the package, has its own set of functions.

- Association rule mining can be implemented using just two functions: `getInputFileForAssociationRules()` and `runAssociationRules()`.
- Frequent pattern mining can be implemented using just two functions: `getInputFileForFrequentPatterns()` and `runFrequentPatterns()`. Additionally, the `getIdDataFrame()` provides a data frame object that indicates the presence or not of a sequence.

3) For use in other OHDSI HADES packages, eg FeatureExtraction: use `createFrequentPatternsCovariateSettings()` to create the `covariateSettings` object. By running `getDbcovariateData()` all extracted FPs are included in a `covariateData` object. Also, at the moment Frequent Patterns can be used with `getPlpData()` but caution should be addressed on the workflow. An example will b soon published where it discusses the current implementations and possible alternatives that should be considered.

Notes:
- Input files should be .txt files.
- None of the functions are exported at the moment. Make sure to use `devtools:load_all()` after loading the package.
- For quick examples have a look at the Preparing the data for mining and Mining sections below.

2 Introduction

This vignette is designed in a way that briefly introduces the advanced user to the AssociationRuleMining R package. By advanced user, we refer to someone already familiar to the OHDSI framework as a start. The package is designed to provide a pipeline for Association Rule Mining (ARM) and Frequent Pattern Mining (FPM) against the OMOP-CDM, even though with an acceptable data input format it can be used to generate results for any general problem or use case.

The package makes use of the open source SPMF library, developed and maintained by Philippe Fournier-Viger. SPMF is a Data Mining Java library implementing a large collection of algorithms related to ARM and FPM, as well as, clustering algorithms and time series mining. The reader is encouraged to have a look at the website of SPMF and explore the documentation and the vast list of implementations offered by the library.

Finally, we would like to encourage the user to experiment with the package and provide feedback, either through the GitHub issue tracker or by email. Have in mind that the package is in the development phase and some features may break. Also, documentation about the conceptual frameworks of ARM and FPM, as well as, documentation related to the algorithms are in production. Interested users may want to have a regular look at the github site of the package for updates. Happy mining!

3 Implementation

3.1 Data Preparation

3.1.1 Getting the necessary data out of the CDM

We assume that an appropriate cohort has been generated for which we would like to extract relevant covariates (e.g. first instance of Myocardial Infarction) and is living on an SQL table. The cohort needs to have the three least usual columns of subject_id, cohort_start_date, and cohort_definition_id. The FeatureExtraction package can then be used to generate temporal covariates of interest for the cohort. For detailed instructions of how to do that have a look at the FeatureExtraction package.

A simple pipeline for generating covariates to be used for ARM is the following:

```
#### Feature Extraction ####
covariateSettings <- createCovariateSettings(useConditionOccurrenceAnyTimePrior = TRUE)

covariateData <- getDbCovariateData(connection = connection,
                                    cdmDatabaseSchema = cdmdatabaseschema, #The database schema where t
                                    cohortDatabaseSchema = resultsDatabaseSchema, #The database schema
                                    cohortTable = "diagnoses", #Name of the cohort table
                                    rowIdField = "subject_id",
                                    covariateSettings = covariateSettings,
                                    cohortTableIsTemp = TRUE) #If the cohort table is temporary or not
```

A simple pipeline for generating temporal covariates to be used for FPM is the following:

```
TemporalcovariateSettings <- createTemporalCovariateSettings(useConditionOccurrence = TRUE,
                                                              temporalStartDays = seq(-(99*365), -1, by = 2) ,
                                                              temporalEndDays = seq(-(99*365)+1, 0, by = 2))

# Extract covariates
TemporalcovariateData<- getDbCovariateData(connection = connection,
```

```
                                        cdmDatabaseSchema = cdmdatabaseschema, #The database schema
                                        cohortDatabaseSchema = resultsdatabaseschema, #The database
                                        cohortTable = cohorttable, #Name of the cohort table
                                        rowIdField = "subject_id",
                                        covariateSettings = TemporalcovariateSettings,
                                        cohortTableIsTemp = TRUE) #If the cohort table is temporary
```

3.1.2 Preparing the data for mining

To generate the appropriate input file for ARM, one should make use of the function `getInputFileForAssociationRules()`. The analogous function for FPM is `getInputFileForFrequentPatterns()`. These functions generate the necessary input files that fulfill the adequate input structure for the algorithms to be implemented. A word of caveat is that the input format has to be a .txt file. Have a look at the SPMF documentation for more details and acceptable alternatives. However, AssociationRuleMining only supports .txt files as inputs at the moment. As these functions generate a text file that is to be processed by the algorithms, they do not need to be assigned to an R object.

```
getInputFileForAssociationRules(covariateDataObject = covariateData, fileToSave = "AssociationRulesExam
```

Note that the `covariateDataObject` argument takes the covariate data object generated by `FeatureExtarction`. The `fileToSave` argument takes the path of the file where the input data is going to be stored and should end in '.txt'.

Similarly for FPM, the same notes as above hold.

```
FPinput <- getInputFileForFrequentPatterns(covariateDataObject = TemporalcovariateData, fileToSave = "F
```

3.1.3 Mining

We would like to emphasize that there is no documentation yet that describes the arguments that are accepted as inputs to the functions calls. We try to describe them in some detail here.

3.1.3.1 Association Rule Mining To run an analysis of ARM, one needs to call `runAssociationRules()`. The function requires four arguments to be specified:

- `algorithm`: Which algorithm to run, currently only accepting one of "Apriori", "Eclat", "FP-Growth", "Relim" and should be quoted.
- `inputFile`: Location and name of the file generated by `getInputFileForAssociationRules()`.
- `outputFile`: Location and name of the file where the results should be saved. **Should be a .txt file**.
- `minSup`: Minimum support for mined items.

Other arguments are possible to be specified, such as, maxLength, indicating the maximum number of items mined in an itemset. However, this is not applicable to every algorithm and it is not implemented at the moment in this package.

An example can be the following:

```
associationSets <- runAssociationRules(algorithm = "Apriori",
                                       inputFile = "AssociationRulesExample.txt",
                                       outputFile = "AssociationRulesExample_Results.txt",
                                       minsup = 0.5 )
```

The function generates another text file saved as "AssociationRulesExample_Results.txt" in the current working directory. It prints on the screen the number of mined itemsets, the time required, memory used and the location of where the output is stored. It also, transforms the output to an R object and therefore needs to be saved.

3.1.4 Frequent Pattern Mining

Similarly for FPM, one needs to call `runFrequentPatterns()`. The function requires 4 mandatory arguments to be specified, and other arguments related to each algorithm.

- `algorithm`: Which algorithm to run, currently only supporting one of "SPAM", "SPADE", "prefixSpan" and should be quoted.
- `inputFile`: Location and name of the file generated by `getInputFileForFrequentPatterns()`.
- `outputFile`: Location and name of the file where the results should be saved. **Should be a .txt file**.
- `minsup`: Mininum support for mined sequences.

Additionally, some of the algorithms accept several additional arguments:

- `minLength`: The minimum length required for a sequence, defaults to 1.
- `maxLength`: The maximum length allowed for a sequence, defaults to 1000.
- `maxGap`: The maximum gap between two events to be considered in a sequence, defaults to 1000.
- `showID`: If sequence ID should be generated, defaults to FALSE.

An example can be the following:

```
frequentPatterns <- runFrequentPatterns(algorithm = "SPADE",
                                        inputFile = "FrequentPatternsExample.txt",
                                        outputFile = "FrequentPatternsExample_Results.txt",
                                        minsup = 0.5,
                                        showID = TRUE)
```

The function generates another text file saved as "FrequentPatternsExample_Results.txt" in the current working directory. It prints on the screen the number of mined sequences, the time required, memory used and the location of where the output is stored. It also, transforms the output to an R object and therefore needs to be saved.

Additionally for FPM, `getIdDataFrame()` generates a data frame object, indicating the presence or not of sequence for each patient id. It is only applicable when `showID = TRUE` when running `runFrequentPatterns()` and accepts as input the output **`.txt`** file of the previous call.

```
getIdDataFrame(inputFile = "FrequentPatternsExample_Results.txt", objectWithIds = FPinput)
```

4 Use in other packages