vignetteSurvivalTimes

2022-11-10

Contents

first we make sure we have the uptodate package

```
remotes::install_github("GidiusVanDeKamp/PlasmodeSim")
```

```
## cli   (3.3.0 -> 3.4.1) [CRAN]
## vctrs (0.5.0 -> 0.5.1) [CRAN]
##
##   There is a binary version available but the source version is later:
##       binary source needs_compilation
## vctrs  0.5.0  0.5.1              TRUE
##
## package 'cli' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'cli'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\gidiu\AppData\Local\R\win-library\4.2\00LOCK\cli\libs\x64\cli.dll to C:
## \Users\gidiu\AppData\Local\R\win-library\4.2\cli\libs\x64\cli.dll: Permission
## denied
```

```
## Warning: restored 'cli'
```

```
##
## The downloaded binary packages are in
##  C:\Users\gidiu\AppData\Local\Temp\RtmpuUHvje\downloaded_packages
```

```
## Warning in i.p(...): installation of package 'vctrs' had non-zero exit status
```

```
## * checking for file 'C:\Users\gidiu\AppData\Local\Temp\RtmpuUHvje\remotes14245cd54292\GidiusVanDeKam
## * preparing 'PlasmodeSim':
## * checking DESCRIPTION meta-information ... OK
## * checking for LF line-endings in source and make files and shell scripts
## * checking for empty or unneeded directories
## Omitted 'LazyData' from DESCRIPTION
## * building 'PlasmodeSim_0.1.0.tar.gz'
##
```

Survival times

In this part we will show how to simulate new survivaltimes. For simulating new censored survival times we need more than one probability, we use the baselinehazard, stored in the a plpModel, we also need a model for the censoring.

Loading the plpData

The first step is to load the data where we will simulate new outcomes for. here we use the package eunomia for a accessting some data. then we pick the settings for the data we want to use, we call this data the plpData.

```r
connectionDetails <- Eunomia::getEunomiaConnectionDetails()

# create the database
Eunomia::createCohorts(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = 'main',
  cohortDatabaseSchema = 'main',
  cohortTable = 'cohort'
)
```

```
## Creating cohort: Celecoxib
##   |                                                                              |
## Creating cohort: Diclofenac
##   |                                                                              |
## Creating cohort: GiBleed
##   |                                                                              |
## Creating cohort: NSAIDs
##   |                                                                              |
## Cohorts created in table main.cohort

##   cohortId       name
## 1        1  Celecoxib
## 2        2 Diclofenac
## 3        3    GiBleed
## 4        4     NSAIDs
##                                                                                        description
## 1    A simplified cohort definition for new users of celecoxib, designed specifically for Eunomia.
## 2    A simplified cohort definition for new users ofdiclofenac, designed specifically for Eunomia.
## 3 A simplified cohort definition for gastrointestinal bleeding, designed specifically for Eunomia.
## 4       A simplified cohort definition for new users of NSAIDs, designed specifically for Eunomia.
##   count
## 1  1844
## 2   850
## 3   479
## 4  2694
```

```r
# ----------------------------------------------------------------------------
# Points PatientLevelPredictionPackage to the Eunomia database
# Tells Eunomia to extract the cohort stored with id = 4 as the target cohort
# and cohort with id = 3 as the outcome cohort. The other settings (...Schema)
# tell the database where to look for the target and the outcome cohorts
# ----------------------------------------------------------------------------
databaseDetails <- PatientLevelPrediction::createDatabaseDetails(
  connectionDetails = connectionDetails,
  cdmDatabaseId = "eunomia",
  cdmDatabaseSchema = 'main',
  cdmDatabaseName = 'Eunomia',
  cohortDatabaseSchema = 'main',
  cohortTable = 'cohort',
  target = 4,
  outcomeDatabaseSchema = 'main',
  outcomeTable = 'cohort',
  outcomeId = 3,
  cdmVersion = 5
)

# Use ?FeatureExtraction::createCovariateSettings to see what the options are
# There are a lot...
covariateSettings <- FeatureExtraction::createCovariateSettings(
  useDemographicsGender = TRUE,
  useDemographicsAgeGroup = TRUE,
  useConditionGroupEraLongTerm = TRUE,
  useDrugGroupEraLongTerm = TRUE,
  endDays = -1,
  longTermStartDays = -365
)

restrictPlpDataSettings <- PatientLevelPrediction::createRestrictPlpDataSettings(
  studyStartDate = '20000101',
  studyEndDate = '20200101',
  firstExposureOnly = TRUE,
  washoutPeriod = 30
)

# issue with studyStartDate/studyEndDate
restrictPlpDataSettings <- PatientLevelPrediction::createRestrictPlpDataSettings(
  firstExposureOnly = TRUE,
  washoutPeriod = 30
)

plpData <- PatientLevelPrediction::getPlpData(
  databaseDetails = databaseDetails,
  covariateSettings = covariateSettings,
  restrictPlpDataSettings = restrictPlpDataSettings
)
```

```
##   |                                                                              |

## Warning: The 'oracleTempSchema' argument is deprecated. Use 'tempEmulationSchema' instead.
## This warning is displayed once every 8 hours.

## Constructing features on server
##   |                                                                              |
## Fetching data from server
## Fetching data took 0.372 secs
```

Defining a training set.

Most of the time we split the dataset into training and a test set. For this purpose we have the function `MakeTraingSet`.

```
populationSettings <- PatientLevelPrediction::createStudyPopulationSettings(
  binary = TRUE,
  includeAllOutcomes = FALSE,
  firstExposureOnly = FALSE,
  washoutPeriod = 180,
  removeSubjectsWithPriorOutcome = FALSE,
  priorOutcomeLookback = 99999,
  requireTimeAtRisk = TRUE,
  minTimeAtRisk = 1,
  riskWindowStart = 1,
  startAnchor = 'cohort start',
  riskWindowEnd = 7300,
  endAnchor = 'cohort start'
)
executeSettings <- PatientLevelPrediction::createExecuteSettings(
  runSplitData = TRUE,
  runSampleData = FALSE,
  runfeatureEngineering = FALSE,
  runPreprocessData = TRUE,
  runModelDevelopment = TRUE,
  runCovariateSummary = TRUE
)
splitSettings <- PatientLevelPrediction::createDefaultSplitSetting(
  testFraction = 0.25,
  trainFraction = 0.75,
  splitSeed = 123,
  nfold = 3,
  type = 'stratified'
)
sampleSettings <- PatientLevelPrediction::createSampleSettings(
  type = 'none'
)
featureEngineeringSettings <-
  PatientLevelPrediction::createFeatureEngineeringSettings(
  type = 'none'
)
preprocessSettings <- PatientLevelPrediction::createPreprocessSettings(
  minFraction = 0,
  normalize = TRUE,
  removeRedundancy = TRUE
)
TrainingSet <- PlasmodeSim::MakeTraingSet(
  plpData = plpData,
  executeSettings = executeSettings,
  populationSettings = populationSettings,
  splitSettings = splitSettings,
  sampleSettings = sampleSettings,
  preprocessSettings = preprocessSettings,
  featureEngineeringSettings = featureEngineeringSettings,
  outcomeId = 3
)
```

```
## Outcome is 0 or 1
## seed: 123
## Creating a 25% test and 75% train (into 3 folds) random stratified split by class
## Data split into 656 test cases and 1974 train cases (658, 658, 658)
## Train Set:
```

```
## Fold 1 658 patients with 120 outcomes - Fold 2 658 patients with 120 outcomes - Fold 3 658 patients w
## 103 covariates in train data
## Test Set:
## 656 patients with 119 outcomes
## Removing 2 redundant covariates
## Normalizing covariates
## Tidying covariates took 0.537 secs
## Train Set:
## Fold 1 658 patients with 120 outcomes - Fold 2 658 patients with 120 outcomes - Fold 3 658 patients w
## 101 covariates in train data
## Test Set:
## 656 patients with 119 outcomes
```

Fitting the model

We pick the desired model by setting the modelsettings. Then we can run the function `fitModelWithCensoring`. This function fits two models one with censoring and one with out the censoring. It stored these models in a list.

```
modelSettings <- PatientLevelPrediction::setCoxModel()

fitCensor <- PlasmodeSim::fitModelWithCensoring(
  Trainingset = TrainingSet$Train,
  modelSettings = modelSettings
)
```

```
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.178 secs
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.128 secs
```

Generating new outcomes times

now that we have our model with the censoring, we can simulate new data. We call the function `simulateSurvivaltimesWithCensoring`

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
  population = TrainingSet$Train$labels,
  populationSettings = populationSettings,
  numberToSimulate = 10
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.186 secs
## Prediction took 0.247 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.18 secs
## Prediction took 0.255 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1   425         6096            0
## 2  1557         7293            0
## 3  2066         2024            0
## 4   664         1329            0
## 5    48         5593            0
## 6   299           18            1
```

```
newdata <- PlasmodeSim::simulateSurvivaltimes(
  plpModel = fitCensor$outcomesModel,
  plpData = plpData,
  numberToSimulate = 10,
  population = TrainingSet$Train$labels,
  populationSettings = populationSettings
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.181 secs
## Prediction took 0.254 secs
```

```
head(newdata)
```

```
##   rowId outcome
## 1  2536    7300
## 2  1882    7300
## 3  1494    7300
## 4  1609    7300
## 5   911    7300
## 6  2610    7300
```

Defining an unfitted model

we can define an model cox by specifying the coefficients and

```
plpModel <- fitCensor$outcomesModel
coeff <- plpModel$model$coefficients
survival <- plpModel$model$baselineSurvival$surv
times <- plpModel$model$baselineSurvival$time

unfittedmodel <- PlasmodeSim::defineCoxModel(
  coefficients = coeff,
  baselinehazard = survival,
  timesofbaselinhazard = times,
  featureEngineering = NULL #  = NULL is the standart setting.
)

newdata <- PlasmodeSim::simulateSurvivaltimes(
  plpModel = unfittedmodel,
  plpData = plpData,
  numberToSimulate = 10,
  population = TrainingSet$Train$labels,
  populationSettings = populationSettings
)
```

```
## Prediction took 0.177 secs
```

```
head(newdata)
```

```
##   rowId outcome
## 1    17      18
## 2  1783    7300
## 3  2600    7300
## 4   964    7300
## 5    30      81
## 6  1182    7300
```

Defining an unfitted model with censoring

```
#we can swap outcomes with censoring.
unfittedcensor<- list(censorModel = unfittedmodel,
                      outcomesModel = fitCensor$outcomesModel)

NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = unfittedcensor,
  plpData = plpData,
  population =  TrainingSet$Train$labels,
  populationSettings = populationSettings,
  numberToSimulate = 200
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.188 secs
## Prediction took 0.261 secs
## Prediction took 0.175 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1  1393           71            0
## 2  1363           36            0
## 3   485         7300            0
## 4   769         7300            0
## 5   244           30            0
## 6   614         7300            0
```

Adjusting the BaselineSurvival

```
adjustedModel <- PlasmodeSim::AdjustBaselineSurvival(
  plpModel = plpModel,
  TrainingSet = TrainingSet$Train,
  plpData = plpData,
  populationSettings = populationSettings,
  timeTofixat = 3592,
  proptofixwith = 0.87,
  intervalSolution= c(-100,100)
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.186 secs
## Prediction took 0.253 secs
```

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = list(censorModel = fitCensor$outcomesModel,
                     outcomesModel = adjustedModel),
  plpData = plpData,
  population =  TrainingSet$Train$labels,
  populationSettings = populationSettings,
  numberToSimulate = 2000
)
```

```
## Prediction took 0.175 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.17 secs
## Prediction took 0.243 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1  2367           13            1
## 2   404           33            1
## 3  1501           14            1
## 4  1545           19            1
## 5  1942            0            1
## 6  1862           18            1
```

Plotting Kaplan Meier estimates

the function `kaplanMeierPlot` visualised the kamplanmeier estimate of a given dataset. It works with ggplot. We can easily compare the simulated data sets with the real dataset by putting them in one plot. For the true data set we set the colour to red.

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
  population = TrainingSet$Train$labels,
  populationSettings = populationSettings,
  numberToSimulate = 1974
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.208 secs
## Prediction took 0.262 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.302 secs
## Prediction took 0.29 secs
```

```
NewOutcomes2 <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
  population = TrainingSet$Train$labels,
  populationSettings = populationSettings,
  numberToSimulate = 1974
)
```

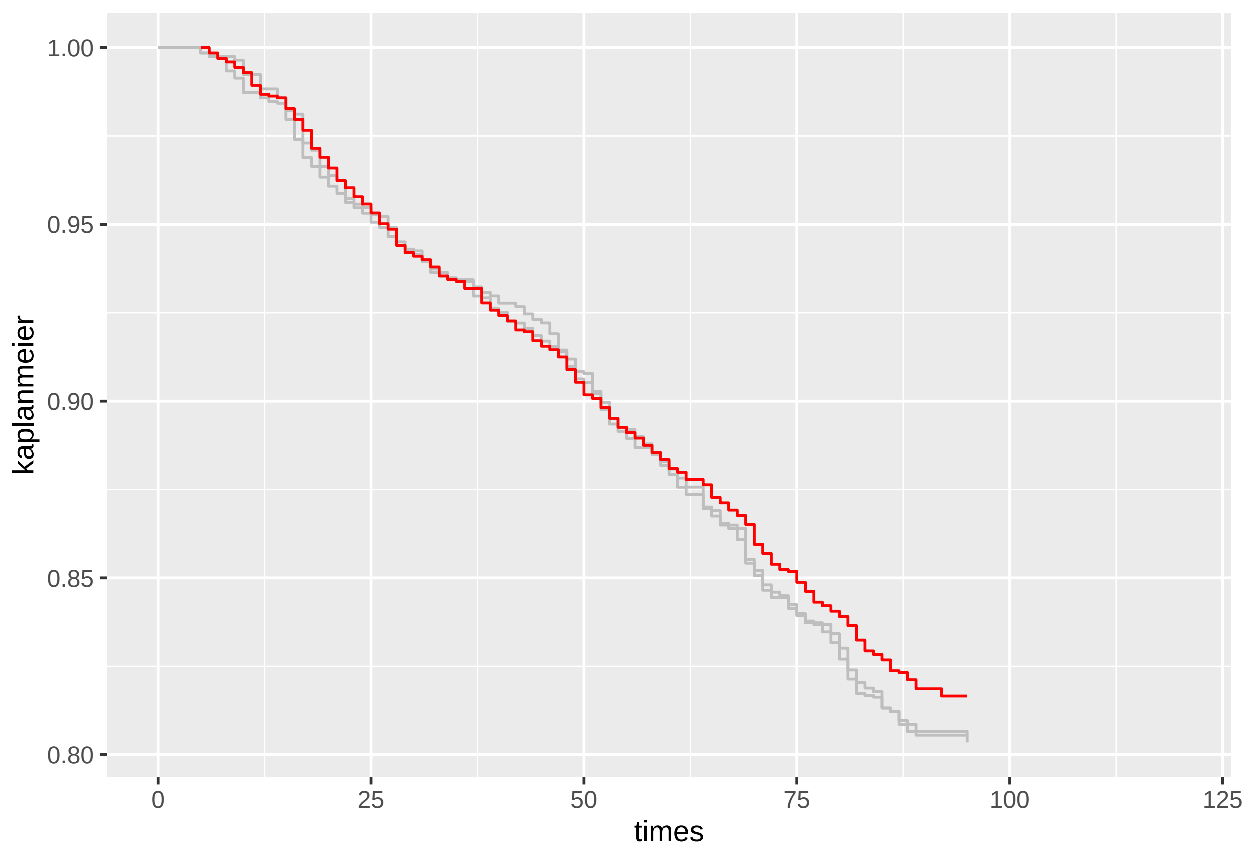
```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.215 secs
## Prediction took 0.274 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.247 secs
## Prediction took 0.299 secs
```

```
ggplot2::ggplot()+
  PlasmodeSim::KaplanMeierPlot( NewOutcomes )+
  PlasmodeSim::KaplanMeierPlot( NewOutcomes2 )+
  PlasmodeSim::KaplanMeierPlot( TrainingSet$Train$labels, colour = 'red' )+
  ggplot2::xlim(c(0,120))
```

```
## Warning: Removed 499 rows containing missing values (‘geom_step()‘).

## Warning: Removed 497 rows containing missing values (‘geom_step()‘).

## Warning: Removed 790 rows containing missing values (‘geom_step()‘).
```



Possible extencions

Here is a list of future extensions to make the package more useful:

- The runPlasmode should have a working analysisId, analysisName and logsettings, like runPlp has.
- one could extend the fitmodel by adding an option for a differen model for the censoring.
- take a look at the feature ingeneering in the `definecoxmodel` function
- add more functions that define unfitted models.

- make it faster by filtering the population only the row ids drawn.