

Vignette PlasmodeSim

2022-10-19

Contents

Installing plasmodeSim using remotes	2
Logistic Regression	3
Setting up	3
Example 1 Simulate from a plpModel	3
Example 2 simulation from unfittedmodel	4
Visual simulations	4
Visual of a specific covariate	6
Survival times	8
Loading the plpData	8
Defining a training set.	10
Fitting the model with censoring	11
Generating new outcomes times	12
Defining an unfitted model	13
Defining an unfitted model with censoring	14
Adjusting the BaselineSurvival	14
Plotting Kaplan Meier estimates	15
runPlasmode	17
Possible extencions	18

Welcome to the vignette about the R package PlasmodeSim. This package is still under development. This package goal is to simulate new outcomes for real patients data. The outcome will follow model you specify.

Installing plasmodeSim using remotes

One can easily install the package using `remotes`, run:

```
install.packages("remotes")  
remotes::install_github("GidiusVanDeKamp/PlasmodeSim")
```

Logistic Regression

Setting up

To start we need a `plpModel` and `plpData`. For information how to obtain these one can look at: <https://ohdsi.github.io/PatientLevelPrediction/articles/BuildingPredictiveModels.html> In this documents we load them from a save file:

```
plpResultLogistic <- PatientLevelPrediction::loadPlpResult( "yourpathForPlpResult")
plpData <- PatientLevelPrediction::loadPlpData( "yourPathForPlpData" )
```

Example 1 Simulate from a plpModel

In this example we obtain new outcomes following a fitted logistic model. We start from a `plpModel`, then run `predictPlp`. At last we generate new outcomes with the function `newOutcomes` that uses the `plpPrediction`.

```
plpModelLog <- plpResultLogistic$model

plpPrediction <- PatientLevelPrediction::predictPlp(
  plpModel = plpModelLog,
  plpData = plpData,
  population = plpData$cohorts
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.206 secs
## Prediction took 0.198 secs
```

When running the function `predictPlp` it returns some information.

```
newOut <- PlasmodeSim::newOutcomes(
  noPersons = 200,
  props = plpPrediction
)
head(newOut)
```

```
##   rowId outcomeCount
## 1     9             0
## 2    26             0
## 3    28             0
## 4    38             0
## 5    43             0
## 6    46             1
```

The `rowId` in the output of `newOutcomes` are the `rowId` of patients that are drawn randomly with the same probability, the patients could be drawn multiple times. If a `rowId` happens to be in the output twice they can have a different outcome. The function `newOutcomes` needs a data set that contains the columns `rowId` and `value`. The column called `value` contains the probabilities used in generating the new outcomes.

Example 2 simulation from unfittedmodel

We here we show how to simulate outcomes from an unfitted logistic model. We use the function `makeLogisiticModel` to specify a logistic model.

```
Parameters <- plpModelLog$model$coefficients
UnfittedParameters <- Parameters
UnfittedParameters[1,1] <- -0.4
UnfittedParameters[3:5,1] <- 0.4
head(UnfittedParameters)
```

```
##      betas covariateIds
## 1  -0.4   (Intercept)
## 2   0.0           6003
## 3   0.4           8003
## 4   0.4           9003
## 5   0.4          8507001
## 6   0.0          28060210
```

For the logistic model it is necessary that the parameters are stored in a dataset with a column called `betas` and a column called `covariateIds`. The function `makeLogisiticModel` makes a `plpModel` from the specified parameters.

```
plpModelunfitted <- PlasmodeSim::makeLogisticModel(UnfittedParameters)
newprobs <- PatientLevelPrediction::predictPlp(
  plpModel = plpModelunfitted,
  plpData = plpData,
  population = plpData$cohorts
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.182 secs
## Prediction took 0.185 secs
```

```
newOut <- PlasmodeSim::newOutcomes(
  noPersons = 2000,
  props = newprobs
)
head(newOut)
```

```
##      rowId outcomeCount
## 1         1             1
## 2         3             0
## 3         4             0
## 4         6             1
## 5         9             1
## 6        10             0
```

Visual simulations

The function `visualOutcome` simulates new data and then plots the frequency of the outcome. Right now the function `visualOutcome` only works for a logistic model. The green line in the plots is the average outcome in the original dataset.

```

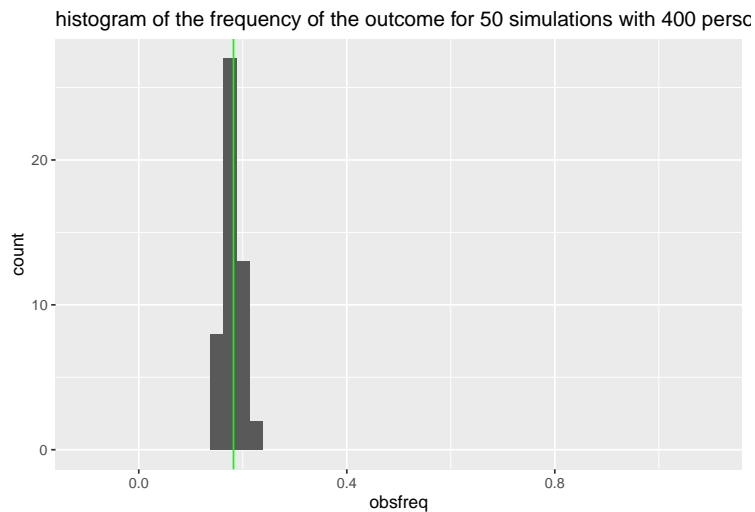
PlasmodeSim::visualOutcome(
  plpData = plpData,
  noSimulations = 50,
  noPersons = 400,
  parameters = Parameters
)

```

```

## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.188 secs
## Prediction took 0.188 secs

```



```

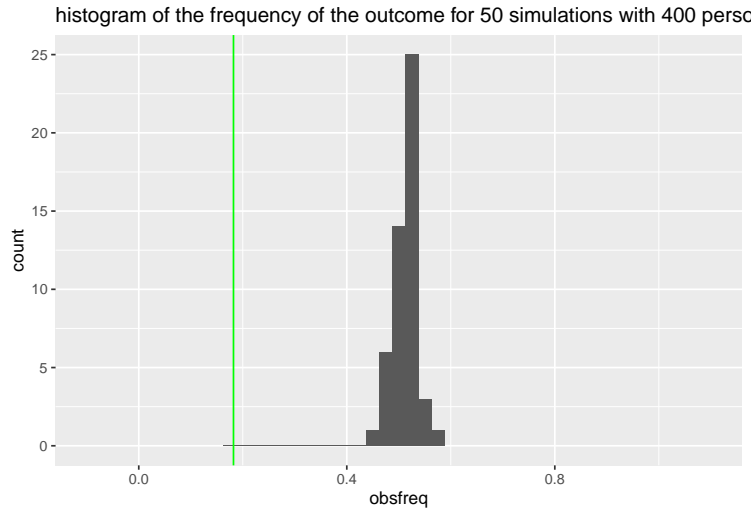
PlasmodeSim::visualOutcome(
  plpData = plpData,
  noSimulations = 50,
  noPersons = 400,
  parameters = UnfittedParameters
)

```

```

## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.24 secs
## Prediction took 0.184 secs

```



Here we have plotted 50 times the frequency of the outcome for a simulated dataset with 200 people. We can see that the outcome count for the fitted parameters is similar as in the original dataset, but when changing the parameters the outcome count also changes.

Visual of a specific covariate

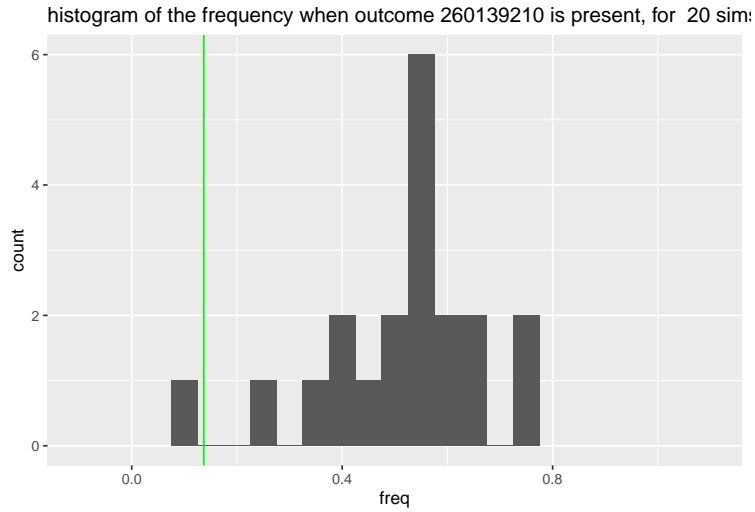
Say we are interested in the outcomes of a group with a specific covariate. Here we picked the third covariate in the model to visualise.

```
covariateIdToStudy<- plpResultLogistic$covariateSummary$covariateId[4]
UnfittedParameters[4,]
```

```
##      betas covariateIds
## 4      0.4           9003
```

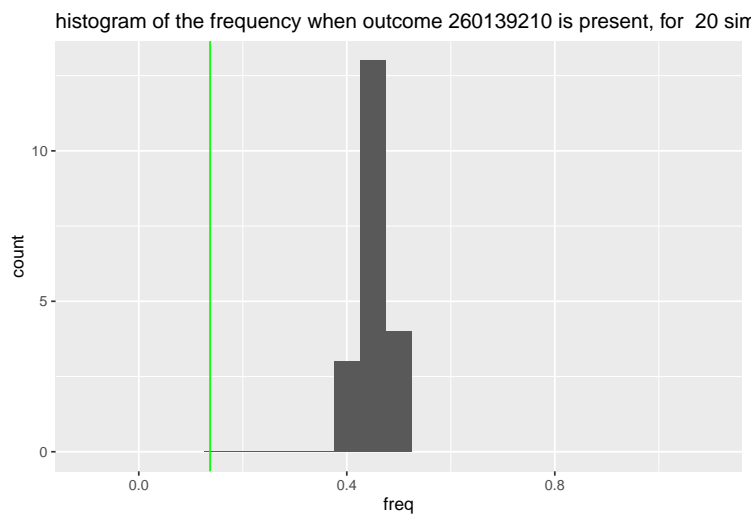
```
PlasmodeSim::visualOutcomeCovariateId(
  plpData=plpData,
  studyCovariateId= covariateIdToStudy,
  noSimulations = 20,
  noPersons = 200,
  parameters= UnfittedParameters
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.233 secs
## Prediction took 0.221 secs
```



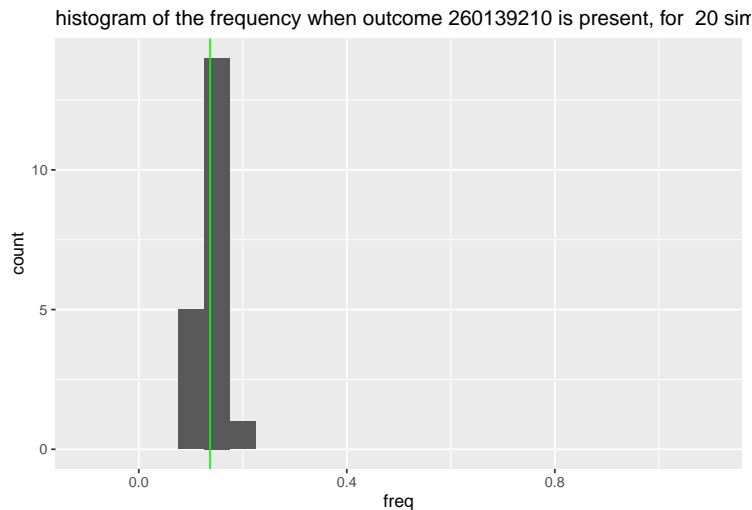
```
PlasmodeSim::visualOutcomeCovariateId2(
  plpData=plpData,
  restrictToCovariateId= covariateIdToStudy,
  noSimulations = 20,
  noPersons= 200,
  parameters= UnfittedParameters
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.191 secs
## Prediction took 0.181 secs
```



```
PlasmodeSim::visualOutcomeCovariateId2(
  plpData=plpData,
  restrictToCovariateId= covariateIdToStudy,
  noSimulations = 20,
  noPersons= 200,
  parameters= Parameters
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.19 secs
## Prediction took 0.182 secs
```



As one can see `visualOutcomeCovariateId` and `visualOutcomeCovariateId2` are very similar, they both calculate and plot the frequency for a group with a specific covariate present. The small difference is that `visualOutcomeCovariateId` filters a newly simulated dataset set to only keep the patients where the covariate is present, and `visualOutcomeCovariateId2` only simulates new outcomes for patients that have the covariate present. We see they are almost identical only `visualOutcomeCovariateId2` is spread out less because the groups for calculating the frequency with are larger. Again we see that when picking the fitted parameters the outcome count for patients with a specific covariate is similar as it was in the original data set.

Survival times

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

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 accessing some data set.

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

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 of diclofenac, 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

```

```

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
)

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
)

```

```
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 0.185 secs
```

Defining a training set.

Most of the time we split the dataset into training and a test set. In order to prepare the data for fitting the model we have the function `MakeTraingSet`. What copies features of the function `patientLevelPrediction::runPlp`. In order to run it we have to create our settings: `populationSettings`, `executeSettings`, `splitSettings`, `sampleSettings`, `featureEngineeringSettings`, `preprocessSettings`. besides all these settings it also needs the `plpData` and the `outcomeId`.

```
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 0.523 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 with censoring

We pick the desired model by setting the `modelsettings`. Then we can run the function `fitModelWithCensoring`. This function fits two `plpModels` one for the censoring and one for outcomes, both of the type specified in the `modelsettings`. It stores these `plpModels` 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 0.151 secs
## Running Cyclops
## Done.
## GLM fit status: OK
## Creating variable importance data frame
## Prediction took 0.135 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 0.178 secs
## Prediction took 0.26 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.169 secs
## Prediction took 0.259 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

```

Since the censoring model Stores to models as a list one can easily generate unsensored outcomes by doing:

```

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 0.169 secs
## Prediction took 0.257 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

Just as before we can define a model that has not been fitted to the data. We specify a cox model by specifying the coefficients/parameters and the baseline survival function.

```

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 standard setting.
)

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

```

```
## Prediction took 0.18 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

There is no function to define an unfitted model with censoring. That is because this can be done easily by making to cox models and storing them in a list. This elements in this list should have the names `censorModel` and `outcomeModel`.

```
#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 0.173 secs
## Prediction took 0.253 secs
## Prediction took 0.181 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

If one want to get a grip on the outcome count on a specific time one can call the function `adjustBaselineSurvival`. This changes the base line function of a model such that, for the training data at the specified time the outcome rate is a specified probability. Since this function solves an equation it needs an specified interval to find this solution.

```
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 0.185 secs
## Prediction took 0.251 secs
```

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

```
## Prediction took 0.177 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.18 secs
## Prediction took 0.254 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 0.167 secs
## Prediction took 0.254 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.168 secs
## Prediction took 0.266 secs
```

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

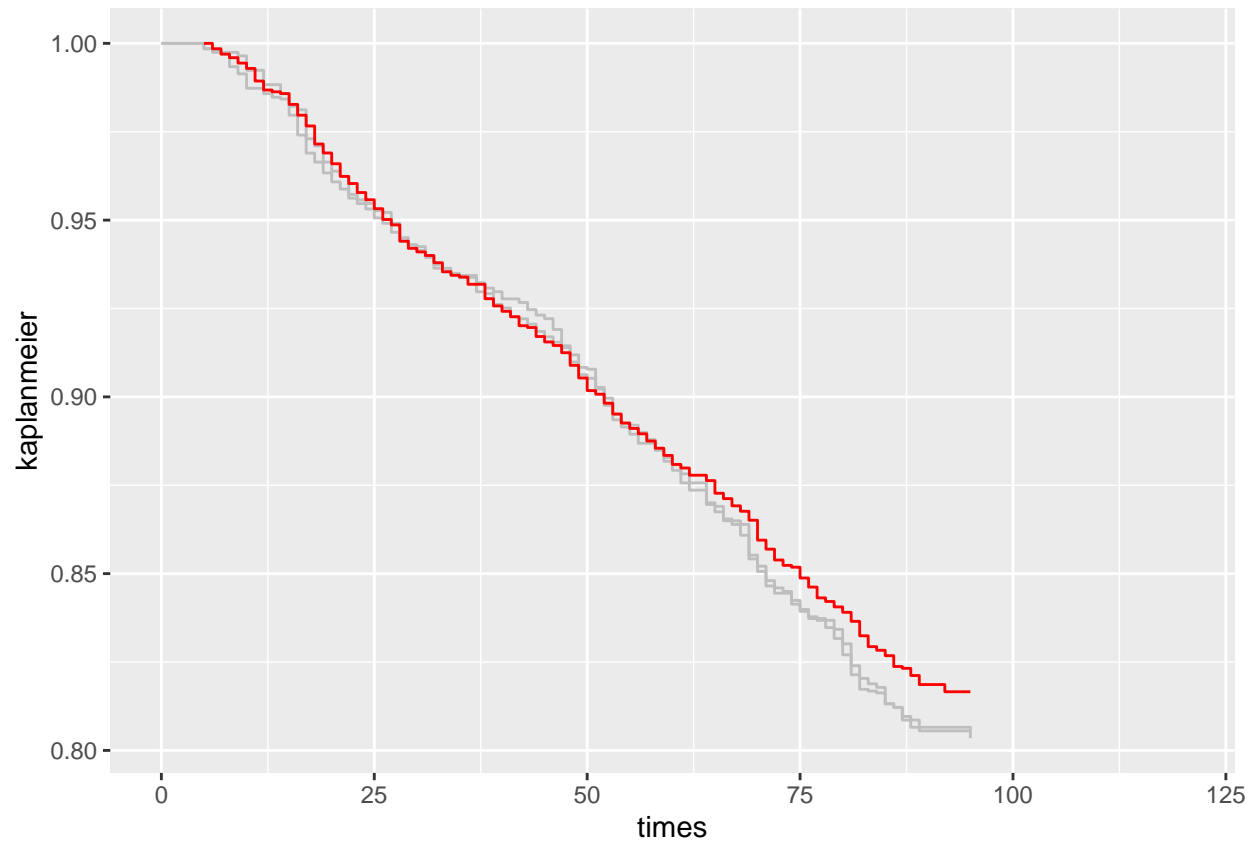
```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.208 secs
## Prediction took 0.265 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.178 secs
## Prediction took 0.271 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()').
```

runPlasmode

the function runPlasmode returns some new simulated survivaltimes, from a model it fits.

```
runPlas <- PlasmodeSim::runPlasmode(
  plpData = plpData,
  outcomeId = 3,
  populationSettings = populationSettings,
  splitSettings = splitSettings,
  sampleSettings = sampleSettings,
  featureEngineeringSettings = featureEngineeringSettings,
  preprocessSettings = preprocessSettings,
  modelSettings = modelSettings,
  executeSettings = executeSettings,
  numberToSimulate = 5
)
```

```
## 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 with 120 outcomes
```

```
## 103 covariates in train data
```

```
## Test Set:
```

```
## 656 patients with 119 outcomes
## Removing 2 redundant covariates
## Normalizing covariates
## Tidying covariates took 0.544 secs
## Train Set:
## Fold 1 658 patients with 120 outcomes - Fold 2 658 patients with 120 outcomes - Fold 3 658 patients
## 101 covariates in train data
## Test Set:
## 656 patients with 119 outcomes
## Running Cyclops
## Done.
## GLM fit status: OK
## Creating variable importance data frame
## Prediction took 0.158 secs
## Running Cyclops
## Done.
## GLM fit status: OK
## Creating variable importance data frame
## Prediction took 0.147 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.183 secs
## Prediction took 0.271 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.209 secs
## Prediction took 0.292 secs
```

```
runPlas
```

```
##   rowId survivalTime outcomeCount
## 1   572          5484             0
## 2  1726          6611             0
## 3   419          7111             0
## 4   522           27             1
## 5   425          7293             0
```

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 models 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 on the row ids drawn, before making their outcomes.