vignette PlasmodeSim

2022-10-19

Welcome to the vignette about the R package PlasmodeSim. This package is still under development.

installing plasmodeSim using remotes

To install using `remotes` run:

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

Setting up

This documents skips some parts, we have skipped the steps to obtain the plpResults and the plpData.

We begin by loading the package, together with the package `PatientLevelPrediction`, since the package `PlasmodeSim` is designed to be used with `PatientLevelPrediction`.

```
library(dplyr)
library(PlasmodeSim)
library(PatientLevelPrediction)

modelSettings <- PatientLevelPrediction::setLassoLogisticRegression()

plpResultLogistic <- PatientLevelPrediction::loadPlpResult("~/R/internshipErasmusMC/simulate-new-patien

plpData<- PatientLevelPrediction::loadPlpData("~/R/internshipErasmusMC/simulate-new-patients-outcomes/pl
```

In this file we will show which functions are in the `PatientLevelPrediction` package, by adding `PatientLevelPrediction::` before the function.

Example 1

In this example we obtain new outcomes of a fitted logistic model.

```
plpModelLog <- plpResultLogistic$model
probabilites <- newPropsParametersPlpModel(plpModelLog, plpData, plpData$cohorts)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.191 secs
## Prediction took 0.177 secs
```

The function predictPlp returned this information.

```
newOut <- newOutcomes(200, probabilites)
head(newOut)
```

```
##   rowId newOutcomes
## 1  2618           0
## 2   801           0
## 3   225           0
## 4   484           0
## 5  1962           0
## 6  2389           0
```

In the output of newOut patients are drawed randomly with the same chance, the patients could be drawed multiple times. If this happens they can have a different outcome. The function `newOutcomes` needs a data set where the column that contains the probabilities is called `value`.

Example 2

We here we show how to simulate new outcomes from an unfitted logistic model.

```
Parameters <- plpModelLog$model$coefficients
UnfittedParameters <- Parameters
UnfittedParameters[1,1] <- -0.4
UnfittedParameters[2:4,1] <- 0.4
head(UnfittedParameters)
```

```
##     betas covariateIds
## 1 -0.4000  (Intercept)
## 2  0.4000         6003
## 3  0.4000         8003
## 4  0.4000         9003
## 5  0.0226      8507001
## 6  0.0000     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`.

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

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.183 secs
## Prediction took 0.163 secs
```

```
newOut <- newOutcomes(200, newprobs)
head(newOut)
```

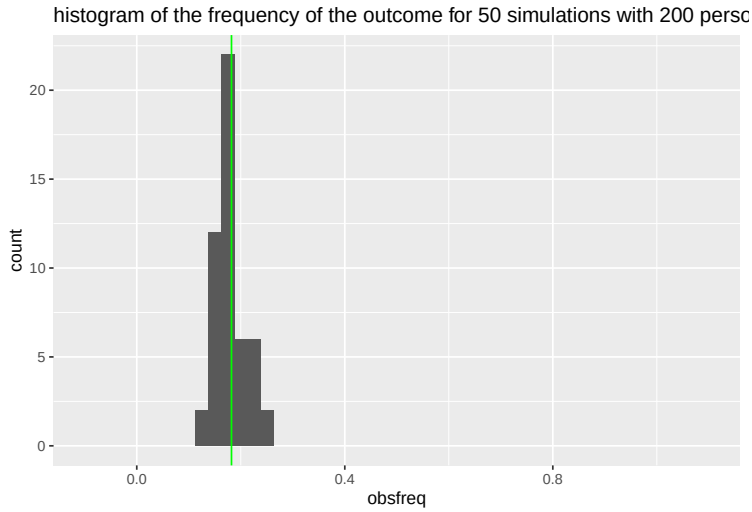
```
##   rowId newOutcomes
## 1  2586           0
## 2    16           0
## 3  2576           0
## 4   457           1
## 5  1696           1
## 6  1298           0
```

Visual simulations

The function `visualOutcome` simulated 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.

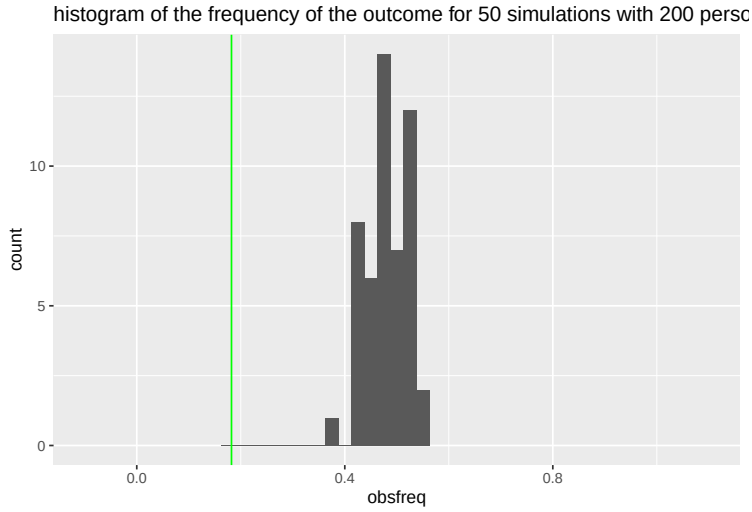
```
visualOutcome(plpData,50,200,Parameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.169 secs
## Prediction took 0.18 secs
```




```
visualOutcome(plpData,50,200,UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.187 secs
## Prediction took 0.179 secs
```



Here we have plotted 50 times the frequency of the outcome for a simulated dataset with 200 people.

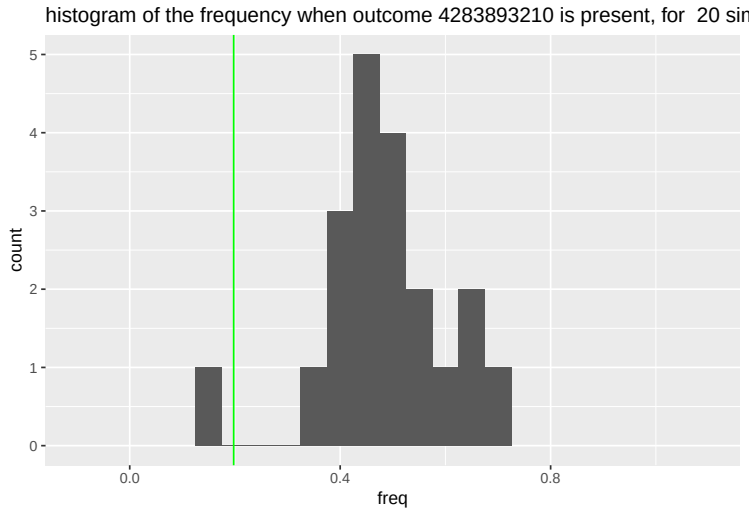
Visual of a specific covariate

```
covariateIdToStudy<- plpResultLogistic$covariateSummary$covariateId[3]
UnfittedParameters[3,]
```

```
##   betas covariateIds
## 3   0.4         8003
```

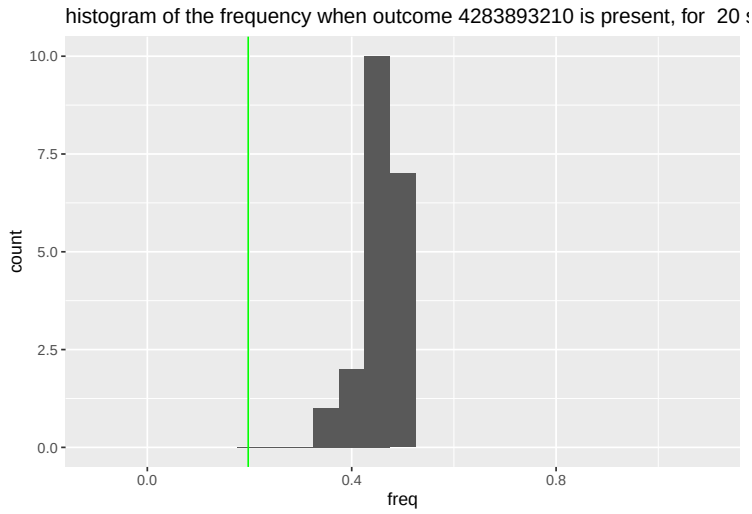
```
visualOutcomeCovariateId(plpData, covariateIdToStudy, 20, 200, UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.193 secs
## Prediction took 0.186 secs
```




```
visualOutcomeCovariateId2(plpData, covariateIdToStudy, 20, 200, UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.174 secs
## Prediction took 0.173 secs
```



As one can see `visualOutcomeCovariateId` and `visualOutcomeCovariateId2` are very similiar, 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 the identical only `visualOutcomeCovariateId2` is spread out less because the groups for calculating the frequency with are bigger.

survival times outcomes.

For simulating new survival times we need more than one probability, we use the baselinehazard, stored in the plpModel. Now there is a function called `expBetaZ` that returns $\exp(\sum_i \beta_i Z_i)$ where β is a vector with the parameters and Z are the covariates of a patients. This could also be included in the function generating new times. (then there would be a posibility that the computer computes things twice, but now maybe it computes expbetaZ's that are never needed).

```
modelSettings <- PatientLevelPrediction::setCoxModel()
plpResult <- PatientLevelPrediction::loadPlpResult("~/R/internshipErasmusMC/simulate-new-patients-outco
plpModel<- plpResult$model
```

```
plpModel$trainDetails$modelName
```

```
## [1] "cox"
```

```
#expbetas <- expBetaZ(plpData, plpModel)
#expbetas<-addExpLp(prediction, baselineSurvival)$exp_lp

#NewOut <- newOutcomesSurvivalTimes(plpModel, expbetas, 20)
#head(NewOut)
```

just as before we have a function to create a dataset that works with the functions.

```
# newcoefficients <- plpModel$model$coefficients
# newcoefficients[1:3,1]<- 0.4
#
# baselineHazard           <- plpModel$model$baselineHazard$surv[1:45]
# timesinthebaselinehazard <- plpModel$model$baselineHazard$time[1:45]
# tail(timesinthebaselinehazard)
#
# plpChangedCoxModel <- makeCoxModel(newcoefficients, baselineHazard, timesinthebaselinehazard)
# newexpbetas <- expBetaZ(plpData, plpChangedCoxModel)
#
# NewOut <- newOutcomesSurvivalTimes(plpChangedCoxModel, newexpbetas, 20)
# head(NewOut)
```

To simulated censored data:

```
# plpModelCensoring <-  plpModel #here should be a function that returns the correct estimate
# expbetasCensor <-  expbetas #here should be a function that returns the correct values
#
```

```
# newCensoredTimes <- newOutcomesCensoredSurvivalTimes(plpModelCensoring, expbetasCensor,
#                                                      plpModel, expbetas, 10)
# head(newCensoredTimes)
```

adjusting the baseline

often it is desired to have the same event rate in the newly simulated data as in the original dataset. to do this we have the function `AdjustBaselineSurvival` with this function one can change the baselineSurvival function such that on time `t` the probability is `p`. Because the function `AdjustbaselineSurvival` solves an equation it needs an interval for finding the solution. one can set the event rate

```
# AdjustedBaseline <- AdjustBaselineSurvival(plpModel$model$baselineHazard,
#                                            66, 1/2, newexpbetas, intervalSolution = c(-100,100))
#
# AdjustedCoxModel <- makeCoxModel(newcoefficients, AdjustedBaseline$surv, AdjustedBaseline$time)
#
# NewOut <- newOutcomesSurvivalTimes(AdjustedCoxModel, newexpbetas, 20)
#
# head(NewOut)
```

notes

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

hoi

```
TrainingSet <- MakeTraingSet(plpData, executeSettings, populationSettings, splitSettings, sampleSettings
```

```
## 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 0.551 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
```

```
fitcensor<-  fitCensoring(TrainingSet$Train, modelSettings)
```

```
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.133 secs
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.137 secs
```

```
NewOutcomes <- simulateWithCensoring(fitcensor, plpData, populationSettings, 20)
```

```
## Outcome is 0 or 1
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.165 secs
## Prediction took 0.26 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.178 secs
## Prediction took 0.266 secs
```

```
head(NewOutcomes)
```

```
##   rowId outcomeCount outcomes
## 1  1290            1       57
## 2   500            1        7
## 3  1864            1       26
## 4   428            0      185
## 5   315            0     7300
## 6   392            0     5658
```

```
# makeCensoringModel(coefficients1, coeffcients2,two baselineHazard) -> returns two models
# adjust only outcome coefficients function.
```

i think it should be something like this: 1. one function that fits and predicts, that only returns the exp lp and the models. 2.

In the end we need 3 things to simulates. 1. baseline hazard function for the outcomes 1. baseline hazard function for the censoring

2. the exponential of the linear prediction for each patient outcome
3. the exponential of the linear prediction for each patient censoring
4. the number of outcomes wanted.