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 0.186 secs
```

The function predictPlp returned this information.

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

```
##   rowId newOutcomes
## 1  2240           0
## 2  2022           0
## 3  1908           0
## 4  2319           1
## 5  2418           0
## 6   776           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 0.167 secs
## Prediction took 0.167 secs
```

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

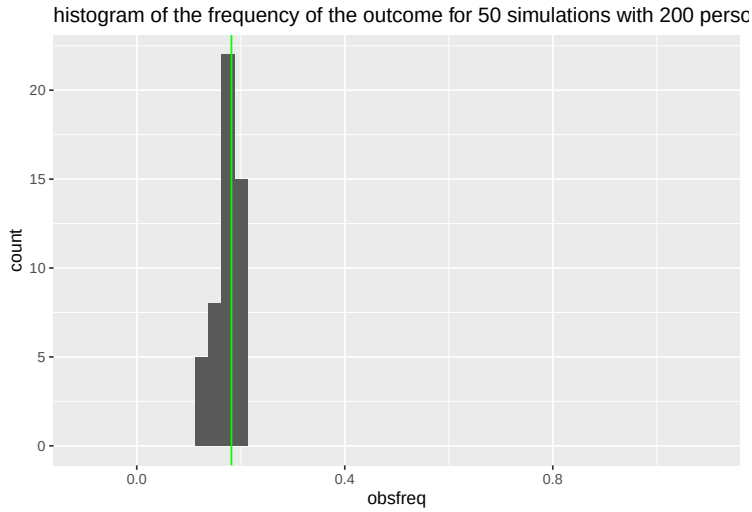
```
##   rowId newOutcomes
## 1   673           0
## 2  1597           1
## 3  1750           0
## 4   222           1
## 5  2352           0
## 6  2144           1
```

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.

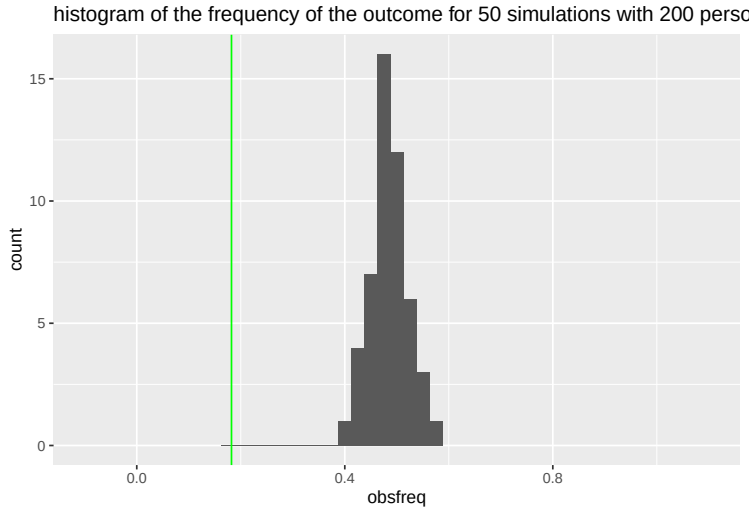
```
visualOutcome(plpData,50,200,Parameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.169 secs
## Prediction took 0.171 secs
```




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

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.181 secs
## Prediction took 0.171 secs
```



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

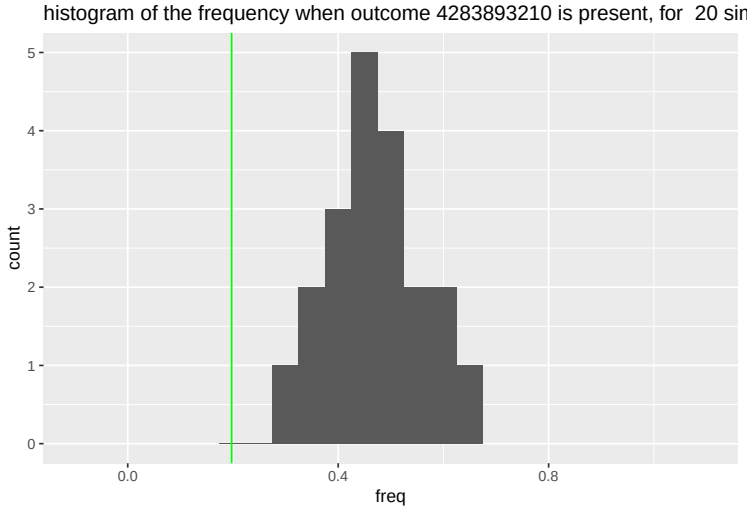
Visual of a specific covariate

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

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

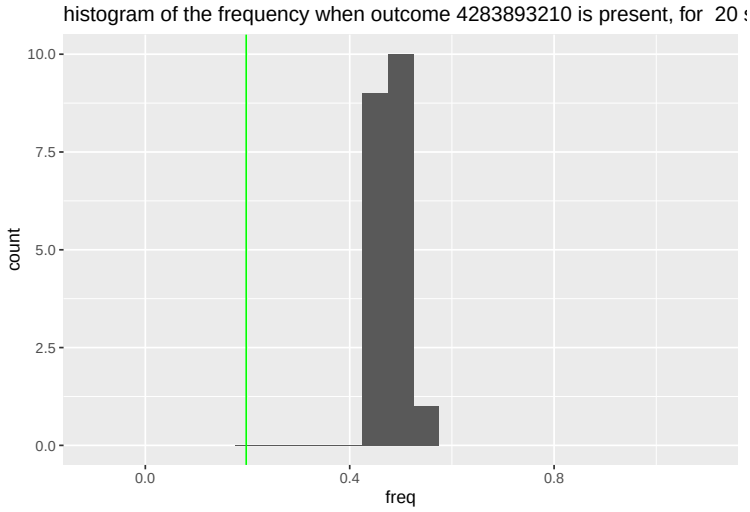
```
visualOutcomeCovariateId(plpData, covariateIdToStudy, 20, 200, UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.203 secs
## Prediction took 0.179 secs
```




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

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.187 secs
## Prediction took 0.176 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-outcor
plpModel<- plpResult$model
```

```
plpModel$trainDetails$modelName
```

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

```
expbetas <- expBetaZ(plpData, plpModel)

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

```
##   outcome rowId
## 1     365  1427
## 2     365  1214
## 3     365   183
## 4      14  2213
## 5     365    84
## 6     365   877
```

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)
```

```
## [1] 47 48 49 50 51 52
```

```
plpChangedCoxModel <- makeCoxModel(newcoefficients, baselineHazard, timesinthebaselinehazard)
newexpbetas <- expBetaZ(plpData, plpChangedCoxModel)

NewOut <- newOutcomesSurvivalTimes(plpChangedCoxModel, newexpbetas, 20)
head(NewOut)
```

```
##   outcome rowId
## 1      52  1526
## 2      52   851
## 3      52   687
## 4      52  1312
## 5      52  2574
## 6      45   196
```

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)
```

```
##   rowId     event outcomes
## 1  1067   outcome       45
## 2  1345   outcome       19
## 3  2046   outcome       82
## 4   690 censoring      365
## 5  1663 censoring       37
## 6   799 censoring       25
```

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.

```
AdjusBaseline <- AdjustBaselineSurvival(plpModel$model$baselineHazard,
                                        66, 1/2, newexpbetas, intervalSolution = c(-100,100))

AdjustedCoxModel <- makeCoxModel(newcoefficients, AdjusBaseline$surv, AdjusBaseline$time)

NewOut <- newOutcomesSurvivalTimes(AdjustedCoxModel, newexpbetas, 20)

head(NewOut)
```

```
##   outcome rowId
## 1      87   485
## 2     365   226
## 3      13   944
## 4     365   262
## 5      37  1623
## 6     365  2388
```