Adding Custom Feature Engineering Functions

Jenna Reps, Egill Fridgeirsson

2023-04-28

Contents

1 Introduction

This vignette describes how you can add your own custom function for feature engineering in the Observational Health Data Sciences and Informatics (OHDSI) `PatientLevelPrediction` package. This vignette assumes you have read and are comfortable with building single patient level prediction models as described in the `BuildingPredictiveModels` vignette.

We invite you to share your new feature engineering functions with the OHDSI community through our GitHub repository.

2 Feature Engineering Function Code Structure

To make a custom feature engineering function that can be used within PatientLevelPrediction you need to write two different functions. The 'create' function and the 'implement' function.

The 'create' function, e.g., create<FeatureEngineeringFunctionName>, takes the parameters of the feature engineering 'implement' function as input, checks these are valid and outputs these as a list of class 'featureEngineeringSettings' with the 'fun' attribute specifying the 'implement' function to call.

The 'implement' function, e.g., implement<FeatureEngineeringFunctionName>, must take as input:

- `trainData` - a list containing:
  - `covariateData`: the `plpData$covariateData`restricted to the training patients
  - `labels`: a data frame that contain `rowId`(patient identifier) and `outcomeCount` (the class labels)

  - `folds`: a data.frame that contains `rowId` (patient identifier) and `index` (the cross validation fold)

- `featureEngineeringSettings` - the output of your create<FeatureEngineeringFunctionName>

The 'implement' function can then do any manipulation of the `trainData` (adding new features or removing features) but must output a `trainData` object containing the new `covariateData`, `labels` and `folds` for the training data patients.

3 Example

Let's consider the situation where we wish to create an age spline feature. To make this custom feature engineering function we need to write the 'create' and 'implement' R functions.

3.1 Create function

Our age spline feature function will create a new feature using the `plpData$cohorts$ageYear` column. We will implement a restricted cubic spline that requires specifying the number of knots. Therefore, the inputs for this are: `knots` - an integer/double specifying the number of knots.

```
createAgeSpline <- function(
                      knots = 5
                      ){

  # create list of inputs to implement function
  featureEngineeringSettings <- list(
    knots = knots
    )

  # specify the function that will implement the sampling
  attr(featureEngineeringSettings, "fun") <- "implementAgeSplines"

  # make sure the object returned is of class "sampleSettings"
  class(featureEngineeringSettings) <- "featureEngineeringSettings"
  return(featureEngineeringSettings)

}
```

We now need to create the 'implement' function `implementAgeSplines()`

3.2 Implement function

All 'implement' functions must take as input the `trainData` and the `featureEngineeringSettings` (this is the output of the 'create' function). They must return a `trainData` object containing the new `covariateData`, `labels` and `folds`.

In our example, the `createAgeSpline()` will return a list with 'knots'. The `featureEngineeringSettings` therefore contains this.

```
implementAgeSplines <- function(trainData, featureEngineeringSettings, model=NULL) {
  # if there is a model, it means this function is called through applyFeatureengineering, meaning it
  if (is.null(model)) {
    knots <- featureEngineeringSettings$knots
    ageData <- trainData$labels
    y <- ageData$outcomeCount
    X <- ageData$ageYear
    model <- mgcv::gam(
      y ~ s(X, bs='cr', k=knots, m=2)
    )
    newData <- data.frame(
      rowId = ageData$rowId,
      covariateId = 2002,
      covariateValue = model$fitted.values
    )
  }
  else {
    ageData <- trainData$labels
    X <- trainData$labels$ageYear
    y <- ageData$outcomeCount
    newData <- data.frame(y=y, X=X)
    yHat <- predict(model, newData)
    newData <- data.frame(
      rowId = trainData$labels$rowId,
      covariateId = 2002,
      covariateValue = yHat
    )
  }

  # remove existing age if in covariates
  trainData$covariateData$covariates <- trainData$covariateData$covariates |>
    dplyr::filter(!covariateId %in% c(1002))

  # update covRef
  Andromeda::appendToTable(trainData$covariateData$covariateRef,
                           data.frame(covariateId=2002,
                                      covariateName='Cubic restricted age splines',
                                      analysisId=2,
                                      conceptId=2002))

  # update covariates
  Andromeda::appendToTable(trainData$covariateData$covariates, newData)

  featureEngineering <- list(
    funct = 'implementAgeSplines',
    settings = list(
      featureEngineeringSettings = featureEngineeringSettings,
      model = model
    )
  )

  attr(trainData$covariateData, 'metaData')$featureEngineering = listAppend(
    attr(trainData$covariateData, 'metaData')$featureEngineering,
```

```
    featureEngineering
  )

  return(trainData)
}
```

4 Acknowledgments

Considerable work has been dedicated to provide the `PatientLevelPrediction` package.

```
citation("PatientLevelPrediction")
```

```
##
## To cite PatientLevelPrediction in publications use:
##
##   Reps JM, Schuemie MJ, Suchard MA, Ryan PB, Rijnbeek P (2018). "Design and implementation of a stand
##   framework to generate and evaluate patient-level prediction models using observational healthcare d
##   _Journal of the American Medical Informatics Association_, *25*(8), 969-975.
##   <https://doi.org/10.1093/jamia/ocy032>.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {J. M. Reps and M. J. Schuemie and M. A. Suchard and P. B. Ryan and P. Rijnbeek},
##     title = {Design and implementation of a standardized framework to generate and evaluate patient-l
##     journal = {Journal of the American Medical Informatics Association},
##     volume = {25},
##     number = {8},
##     pages = {969-975},
##     year = {2018},
##     url = {https://doi.org/10.1093/jamia/ocy032},
##   }
```

Please reference this paper if you use the PLP Package in your work:

Reps JM, Schuemie MJ, Suchard MA, Ryan PB, Rijnbeek PR. Design and implementation of a standardized framework to generate and evaluate patient-level prediction models using observational healthcare data. J Am Med Inform Assoc. 2018;25(8):969-975.

This work is supported in part through the National Science Foundation grant IIS 1251151.