Patient Level Prediction Examples
version 1.1

Jenna Reps

June 29, 2016

1 Introduction

The patient level prediction package is a R package then enables users to download data from health databases in CDM format, constructs covariates/features and trains various machine learning models to predict an outcome within a user specified cohort during a defined time period.

The users needs to:

a. create an 'at risk' cohort using atlas or raw sql

b. create an 'outcome' cohort using atlas or raw sql

c. define the population of interest (e.g., first time outcome)

d. define the prediction window (e.g., within 1 year of 'at risk' start date)

e. pick a test/train split (e.g. 30% test: 70% train splitting by time)

f. pick a machine learning model (e.g., logistic regression)

The machine learning models currently available are:

- Logistic regression with regularization
- Gradient boosting machines
- Random forest
- K nearest neighbors
- Naive Bayes
- (comming soon) Neural network
- (coming soon) Deep learning

2 Example using Atlas

2.1 Cohort Creation

The patient level prediction package enables you to use the cohorts created in atlas. Go to atlas and create the 'at risk' cohort and the 'outcome' cohort. The url of the cohort will be in the form:

```
http://yourwebsite.com/atlas/#/cohortdefinitions/2262
```

where the number at the end is the cohort_definition_id. Keep a note of this id as it will be required to extract the patient level prediction data.
For the purpose of the example we will assume our at risk cohort has an id of: 2262 and our outcome cohort has an id of: 1913.

2.2 Extracting data

First we need to load the patient level prediction library and set the location where you want to save results:

```
library(PatientLevelPrediction)
wd <- file.path("C:/Users/jreps/Documents", 'patientprediction')
```

Next, set up the database connection:

```
databaseSchema <- 'cdm_optum_v5.dbo'
database <- strsplit(databaseSchema, '\\.')[[1]][1]

dbms <- "pdw"
user <- enterusername
pw <- enterpassword
server <- "JRDUSAPSCTL01"
port <- 17001
oracleTempSchema <- NULL
cdmVersion <- 5

connectionDetails <- DatabaseConnector::createConnectionDetails(dbms =
    dbms,
                                                                server =
                                                                    server,
                                                                user = user,
                                                                password = pw,
                                                                port = port)
```

Then specify the covariates/features that will be used by the prediction algorithm. These are created using the FeatureExtraction package.

```
# covariate settings:
covSettings <- FeatureExtraction::createCovariateSettings(
```

```
useCovariateCohortIdIs1 = FALSE,
useCovariateDemographics = TRUE,
useCovariateDemographicsGender = TRUE,
useCovariateDemographicsRace = TRUE,
useCovariateDemographicsEthnicity = TRUE,
useCovariateDemographicsAge = TRUE,
useCovariateDemographicsYear = TRUE,
useCovariateDemographicsMonth = TRUE,
useCovariateConditionOccurrence = TRUE,
useCovariateConditionOccurrence365d = TRUE,
useCovariateConditionOccurrence30d = FALSE,
useCovariateConditionOccurrenceInpt180d = FALSE,
useCovariateConditionEra = FALSE,
useCovariateConditionEraEver = FALSE,
useCovariateConditionEraOverlap = FALSE,
useCovariateConditionGroup = T,
useCovariateConditionGroupMeddra = T,
useCovariateConditionGroupSnomed = FALSE,
useCovariateDrugExposure = T,
useCovariateDrugExposure365d = T,
useCovariateDrugExposure30d = FALSE,
useCovariateDrugEra = FALSE,
useCovariateDrugEra365d = FALSE,
useCovariateDrugEra30d = FALSE,
useCovariateDrugEraOverlap = FALSE,
useCovariateDrugEraEver = FALSE,
useCovariateDrugGroup = T,
useCovariateProcedureOccurrence = T,
useCovariateProcedureOccurrence365d = T,
useCovariateProcedureOccurrence30d = FALSE,
useCovariateProcedureGroup = T,
useCovariateObservation = T,
useCovariateObservation365d = T,
useCovariateObservation30d = FALSE,
useCovariateObservationCount365d = FALSE,
useCovariateMeasurement = T,
useCovariateMeasurement365d = T,
useCovariateMeasurement30d = FALSE,
useCovariateMeasurementCount365d = FALSE,
useCovariateMeasurementBelow = FALSE,
useCovariateMeasurementAbove = FALSE,
useCovariateConceptCounts = FALSE,
useCovariateRiskScores = T,
useCovariateRiskScoresCharlson = T,
useCovariateRiskScoresDCSI = T,
useCovariateRiskScoresCHADS2 = T,
useCovariateRiskScoresCHADS2VASc = T,
useCovariateInteractionYear = FALSE,
useCovariateInteractionMonth = FALSE,
excludedCovariateConceptIds = c(),
```

```
includedCovariateConceptIds = c(),
deleteCovariatesSmallCount = 50)
```

The final step for extracting the data is to using the getDbPlpData function and input the connection details, the database schema where the atlas cohorts are stored, the cohort_definition_ids for the cohort and outcome, the washoutPeriod is the minimum number of days prior to cohort index date that the person must have been observed to be included into the data and finally input the previously constructed covariate settings.

```
# extract the data:
plpData <- PatientLevelPrediction::getDbPlpData(connectionDetails,
                                                cdmDatabaseSchema=databaseSchema,
                                                cohortId=2292,
                                                                outcomeIds=1913,
                                                outcomeDatabaseSchema =
                                                    databaseSchema,
                                                outcomeTable = 'cohort',
                                                cohortDatabaseSchema =
                                                    databaseSchema,
                                                cohortTable = 'cohort',
                                                cdmVersion=5,
                                                washoutPeriod=365,
                                                covariateSettings=covSettings)
```

Now might be a good time to save, the data extracted is in coo format ().

```
# save the coo format data:
PatientLevelPrediction::savePlpData(plpData, file=file.path(wd,'coo'))

# you can load the sved data using the following code:
plpData <- PatientLevelPrediction::loadPlpData(file=file.path(wd,'coo'))
```

2.3 Create population

Now we have the data downloaded, we need to define the prediction problem by creating the population (define the prediction period, the minimum prior observation and other exclusions):

```
# to create a population to predict outcome with cohort_definition_if of
    1913
# where we only want first time exposure (earliest at risk cohort start
    date per person), a minimum of 365 days prior observstion
    (washoutPeriod) and we wish to do the prediction between the at
    risk cohort start date + 28 day until the cohort_end_date
population <- createStudyPopulation(plpData, outcomeId = 1913,
                                     firstExposureOnly = F,
                                     washoutPeriod = 365,
```

```
                                    removeSubjectsWithPriorOutcome = F,
                                priorOutcomeLookback=0,
                                                riskWindowStart = 1,
                                requireTimeAtRisk = T,
                                                minTimeAtRisk = 180,
                                addExposureDaysToStart = FALSE,
                                                riskWindowEnd = 28,
                                addExposureDaysToEnd=T)
```

2.4 Create model settings

To run a logistic regression with lasso regularization run:

```
# create the logistic regression settings:
lr_model <- PatientLevelPrediction::logisticRegressionModel()
```

To run a gradient boosting model run:

```
# try gradient boosting machine with grid search testing number of trees
    of 100,150
# and 200 and max depth of 3,4,5,6 or 10 levels.
gbm_model <- PatientLevelPrediction::GBMclassifier_xgboost(
                        ntrees=c(100,150,200),
                                    max_depth = c(3,4,5,6,10))
```

2.5 Run model development

```
# now run the developModel function
# inputting the plpData, popualtion, model settings
# as well as the type of evaluation and test/train split fraction and
# number of folds used during during cross validation
lr_results <- PatientLevelPrediction::developModel(population, plpData,
                                                modelSettings = lr_model,
                                                testSplit='time',
                                                testFraction=0.25,
                                                nfold=2)

# for the gradient boosting:
gbm_results <- PatientLevelPrediction::developModel(population, plpData,
                                                 modelSettings = gbm_model,
                                                 testSplit='time',
                                                 testFraction=0.25,
                                                 nfold=2)
```

2.5.1 Converting to lib SVM

The python methods require converting the coo format data into libsvm format (if you want to use random forest, naive bayes or neural networks/deep learning you will need to covert the data:

```
# convert the data and save the libsvm file to the S:/libsvm/data folder
plpData.lsvm <- PatientLevelPrediction::convertToLibsvm(plpData,
    file=file.path('S:/libsvm','data') )

# you can save/load the libsvm data in the same way you save the coo
    format:
PatientLevelPrediction::savePlpData(plpData.lsvm,
    file=file.path(wd,'libsvm'))
plpData.lsvm <-
    PatientLevelPrediction::loadPlpData(file=file.path(wd,'libsvm'))
```

The libsvm data can also be fed into the createStudyPopulation() function the same way as the original data format:

```
population.lsvm <- createStudyPopulation(plpData.lsvm, outcomeId = 1913,
                                 firstExposureOnly = F,
                                      washoutPeriod = 365,

                                          removeSubjectsWithPriorOutcome
                                          = F,
                                      priorOutcomeLookback=0,
                                          riskWindowStart = 1,
                                      requireTimeAtRisk = T,
                                          minTimeAtRisk = 180,
                                      addExposureDaysToStart = FALSE,
                                          riskWindowEnd = 28,
                                      addExposureDaysToEnd=T)
```

this data and population can now be used to train a random forest or naive bayes model using a python back end:

```
# try random forest with grid search testing number of trees of 100,150
# and 200 and mtry is default of -1 (square root of total number of
    features).
rf_model <-
    PatientLevelPrediction::RFclassifier_python(ntrees=c(100,150,200),
    mtry=-1)
rf_results <- PatientLevelPrediction::developModel(population.lsvm,
    plpData.lsvm,
                                                modelSettings = rf_model,
                                                testSplit='time',
                                                testFraction=0.25,
                                                nfold=2),
```

```
# try naive bayes model
# and 200 and mtry is default of -1 (square root of total number of
    features).
nb_model <- PatientLevelPrediction::NBclassifier_python()
nb_results <- PatientLevelPrediction::developModel(population.lsvm,
    plpData.lsvm,
                                                   modelSettings = nb_model,
                                                   testSplit='time',
                                                   testFraction=0.25,
                                                   nfold=2),
```

3 Example using custom cohorts

Occasionally you may need to create your own cohorts (a table containing cohort_definition_id, subject_id, cohort_start_date and cohort_end_date fields). For example, imagine you have a cohort of pregnant women with a cohort_start_date corresponding to the conception date, a cohort_definition_id of 1, the pregnancy cohort table is called 'pregnancy' and it is in the 'writeable.dbo' schema. You also have a cohort corresponding to gestational diabetes with the cohort_start_date being the first gestational diabetes record date, a cohort_definition_id of 1, the gestational diabetes table is 'gdm' and it is in the 'madeup.dbo' schema. To run the patient level prediction package using these cohort tables you need to extract the data by referencing the correct tables and databases (where we remove people who have less that 180 days observation prior to cohort_start_date):

```
# extract the data:
plpData <- PatientLevelPrediction::getDbPlpData(connectionDetails,
                                                cdmDatabaseSchema=databaseSchema,
                                                cohortId=1,
                                                                    outcomeIds=1,
                                                outcomeDatabaseSchema =
                                                    'madeup.dbo',
                                                outcomeTable = 'gdm',
                                                cohortDatabaseSchema =
                                                    'writeable.dbo',
                                                cohortTable = 'pregnancy',
                                                cdmVersion=5,
                                                washoutPeriod=180,
                                                covariateSettings=covSettings)
```

4 Extras

4.1 custom machine learning models

There is the flexibility within the package of adding your own machine learning models. You need to specify the model function, the model settings function and the prediction function.
An example of model setting functions for the default models can be found in the createSettings.R file within the PatientLevelPrediction r package.

4.1.1 model setting function

This function takes as input the hyper-parameter values to construct a grid search and returns the settings ready to be used by the model function. The output of this function is a list containing:

- model - a string specifying the name of the model function (e.g., 'lasso_lr' corresponds to the regularized logistic regression)
- param - a list, with each element containing one of the hyper-parameter combinations to test during the grid search.

the output should be of class 'modelSettings' and have an attribute named libSVM with is binary and specifies whether the model requires libSVM format.

4.1.2 model function

To add your own model you need to create a function that takes as input:

- population
- plpData
- param
- index
- quiet
- outcomeId
- cohortId

Where the population is the study population, the plpData is the plpData, param is a list containing the hyper-parameters combinations that will be searched during model training (this gets created by the model settings function you write)
and the output of this function must be a list containing: itemize

model (the model function name as a string)

modelSettings (a list of the settings input into the model training)

trainCVAuc (the cross validation performance on the train set)

metaData (the plpData metaData)

populationSettings (the settings of the population)

outcomeId (being predicted)

cohortId (which cohort is the prediction run on)

varImp (variable importance)

trainingTime (time it took to train the model)
In addition, the result list should have a class of 'plpModel', and attribute type of 'plp' and an attribute predictionType of 'binary':

```
class(result) <- 'plpModel'
attr(result, 'type') <- 'plp'
attr(result, 'predictionType') <- 'binary'
```

For examples of default models see the defaultModels.R code within the patientLevelPrediction package.

4.1.3 prediction function

The prediction function needs to take as input:

- plpModel (the trained model)
- plpData (patient level prediction format data)
- population (patient level prediction population data.frame)
- silent (binary whether the progress should be reported)

The prediction function should then use these inputs to make a prediction for the population using the plpData and plpModel and returns a data.frame containing 'rowId', 'outcomeCount', 'indexes', 'value' where value is the prediction value between 0 and 1, rowId is the id used to reference a person in the population, indexes is a column indicating whether a rowId was used in the test/train set or the cross validation fold number (when applicable) and outcomeCount is the true label (when applicable). The output should have the same number of rows as the input population. The data.frame should have an attribute of metaData specifying the predictionType (binary/multiclass):

```
attr(prediction, 'metaData') <- list(predictionType='binary')
```

Examples of prediction functions for the defaul models can be found in the Predict.R file within the PatientLevelPrediction package.

5 Machine Learning Methods

Method	Parameters
Lasso Logistic Regression	var (starting variance)
Random forest	mtry (number of features in each tree), ntree (number of trees), max_depth (max levels in tree), min_rows (minimum data points in in node), balance (balance class labels)
Gradient boosting machine	ntree (number of trees), max_depth (max levels in tree), min_rows (minimum data points in in node), learning rate
KNN	k (number of neighbours), weighted (weight by inverse frequency)
Naive Bayes	-