Building patient-level predictive models

Jenna Reps, Martijn J. Schuemie, Patrick B. Ryan, Peter R. Rijnbeek

2017-07-19

Contents

1 Introduction **2**

2 Installation instructions **2**

3 Data extraction **3**
3.1 Configuring the connection to the server . . . 3
3.2 Preparing the cohort and outcome of interest . . . 3
3.3 Extracting the data from the server . . . 6
3.4 Saving the data to file . . . 7

4 Applying additional inclusion criteria **8**

5 Model Development **9**
5.1 Defining the model settings . . . 9
5.2 Model training . . . 9
5.3 Saving and loading . . . 10

6 Model Evaluation **11**
6.1 ROC plot . . . 11
6.2 Calibration plot . . . 11
6.3 Preference distribution plots . . . 11
6.4 Box plots . . . 15
6.5 Test-Train similarity plot . . . 15
6.6 Variable scatter plot . . . 15
6.7 Plot Precision Recall . . . 17
6.8 Demographic Summary plot . . . 19

7 External validation **20**

8 Acknowledgments **21**

1 Introduction

This vignette describes how you can use the `PatientLevelPrediction` package to build patient-level predictive models. The package enables data extraction, model building, and model evaluation using data from databases that are translated into the Observational Medical Outcomes Partnership Common Data Model (OMOP CDM).

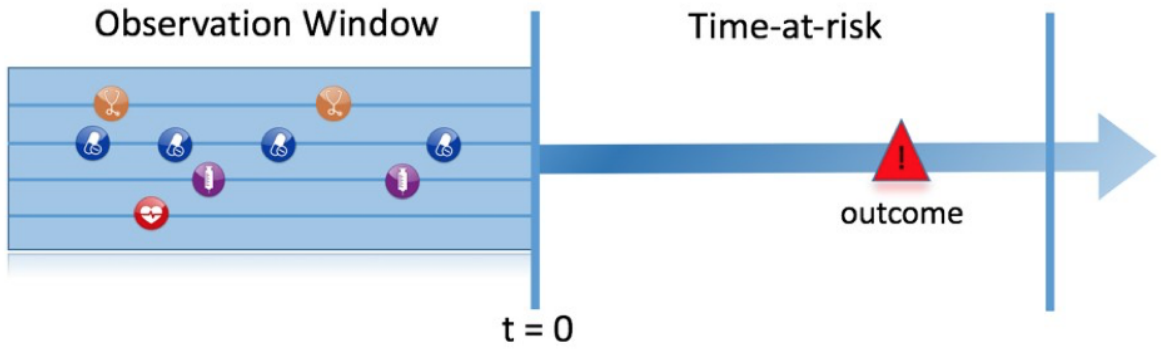


Figure 1: The prediction problem

Figure 1 illustrates the prediction problem we address. Among a population at risk, we aim to predict which patients at a defined moment in time (t = 0) will experience some outcome during a time-at-risk. Prediction is done using only information about the patients in an observation window prior to that moment in time.

To develop a model the user needs to take the following steps:

1. Create the at risk and outcome cohorts
2. Extract the patient-level data from the server
3. Define the population of interest
4. Define the time-at-risk
5. Pick a test/train split
6. Create model settings
7. Fit the model
8. Evaluate the model
9. Apply the model

We have selected the well-studied topic of predicting re-hospitalization to walk you through these steps. The model will be developed for a diabetes type 2 population.

2 Installation instructions

Before installing the `PatientLevelPrediction` package make sure you have Java available. Java can be downloaded from www.java.com. For Windows users, RTools is also necessary. RTools can be downloaded from CRAN.

Furthermore, a python installation is required for some of the machine learning algorithms. We advise to install Python using Anaconda (https://www.continuum.io/downloads)

The `PatientLevelPrediction` package is currently maintained in a GitHub repository (https://github.com/OHDSI/PatientLevelPrediction), and has dependencies on other packages in Github. All of these packages can be downloaded and installed from within R using the `drat` package:

```
install.packages("drat")
drat::addRepo("OHDSI")
install.packages("PatientLevelPrediction")
```

Once installed, you can type `library(PatientLevelPrediction)` to load the package.

3 Data extraction

The `PatientLevelPrediction` package requires longitudinal observational healthcare data in the OMOP Common Data Model format. The user will need to specify two things:

1. Time periods for which we wish to predict the occurrence of an outcome. We will call this the **cohort of interest** or cohort for short. One person can have multiple time periods, but time periods should not overlap.
2. Outcomes for which we wish to build a predictive model.

The first step in running the `PatientLevelPrediction` is extracting all necessary data from the database server holding the data in the CDM.

3.1 Configuring the connection to the server

We need to tell R how to connect to the server where the data are. `PatientLevelPrediction` uses the `DatabaseConnector` package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")

cdmDatabaseSchema <- "my_cdm_data"
cohortsDatabaseSchema <- "my_results"
cdmVersion <- "5"
```

The last three lines define the `cdmDatabaseSchema` and `cohortsDatabaseSchema` variables, as well as the CDM version. We will use these later to tell R where the data in CDM format live, where we want to create the cohorts of interest, and what version CDM is used. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

3.2 Preparing the cohort and outcome of interest

First we need to define the cohort of persons for which we want to perform the prediction and we need to define the outcomes we want to predict.

The cohort and outcomes are provided as data in a table on the server that has the same structure as the 'cohort' table in the OMOP CDM, meaning it should have the following columns:

- `cohort_definition_id`, a unique identifier for distinguishing between different types of cohorts, e.g. cohorts of interest and outcome cohorts.
- `subject_id`, a unique identifier corresponding to the `person_id` in the CDM.
- `cohort_start_date`, the start of the time period where we wish to predict the occurrence of the outcome.

- `cohort_end_date`, which can be used to determine the end of the prediction window. Can be set equal to the `cohort_start_date` for outcomes.

The observational and health data sciences & informatics (OHDSI) community has developed a tool named ATLAS which can be used to create cohorts based on inclusion criteria. We can also write custom SQL statements against the CDM.

3.2.1 Cohort creation using ATLAS

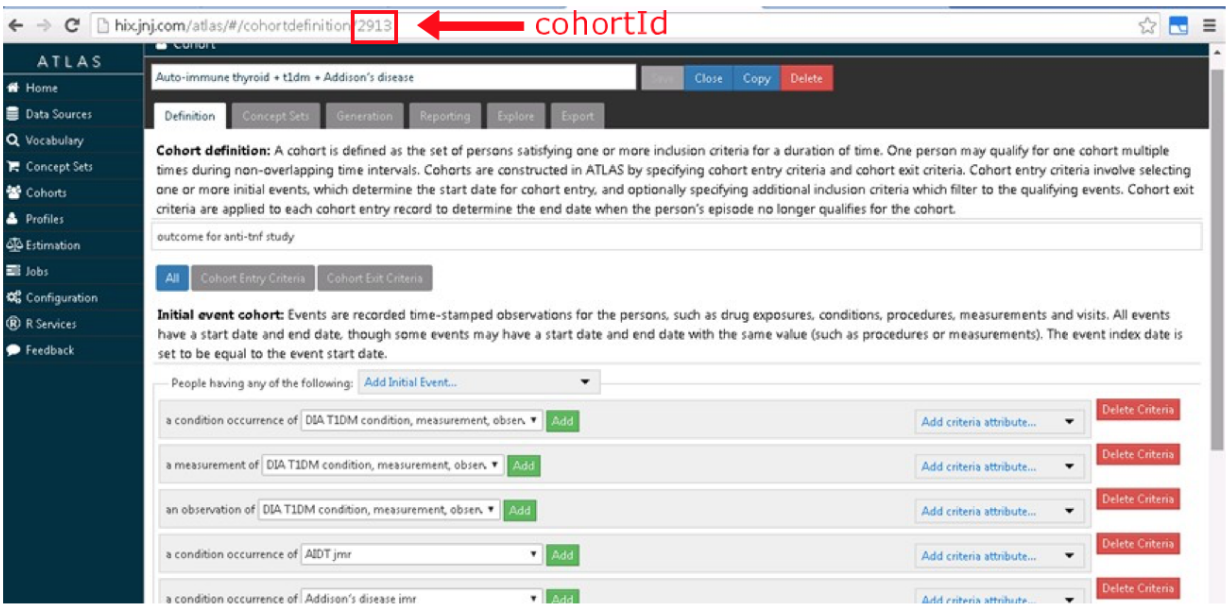


Figure 2: Cohort creation using ATLAS

ATLAS as shown in Figure 2 allows you to define cohorts interactively by specifying cohort entry and cohort exit criteria. Cohort entry criteria involce selecting one or more initial events, which determine the start date for cohort entry, and optionally specifying additional inclusion criteria which filter to the qualifying events. Cohort exit criteria are applied to each cohort entry record to determine the end date when the person's episode no longer qualifies for the cohort. For the outcome cohort the end date is less relevant. More details on the use of ATLAS can be found on the OHDSI wiki pages.

When a cohort is created in ATLAS the cohortid is needed to extract the data in R. The cohortid can be found in the link as shown in Figure 2.

3.2.2 Custom cohort creation

It is also possible to create cohorts without the use of ATLAS. Using custom cohort code (SQL) you can make more advanced cohorts if needed.

For our example study, we need to create the cohort of diabetics that have been hospitalized and have a minimum amount of observation time available before and after the hospitalization. We also need to define re-hospitalizations, which we define as any hospitalizations occurring after the original hospitalization.

For this purpose we have created a file called *HospitalizationCohorts.sql* with the following contents:

```
/**********************************
File HospitalizationCohorts.sql
```

```
**********************************/
IF OBJECT_ID('@resultsDatabaseSchema.rehospitalization', 'U') IS NOT NULL
    DROP TABLE @resultsDatabaseSchema.rehospitalization;

SELECT visit_occurrence.person_id AS subject_id,
    MIN(visit_start_date) AS cohort_start_date,
    DATEADD(DAY, @post_time, MIN(visit_start_date)) AS cohort_end_date,
    1 AS cohort_definition_id
INTO @resultsDatabaseSchema.rehospitalization
FROM @cdmDatabaseSchema.visit_occurrence
INNER JOIN @cdmDatabaseSchema.observation_period
    ON visit_occurrence.person_id = observation_period.person_id
INNER JOIN @cdmDatabaseSchema.condition_occurrence
ON condition_occurrence.person_id = visit_occurrence.person_id
WHERE visit_concept_id IN (9201, 9203)
    AND DATEDIFF(DAY, observation_period_start_date, visit_start_date) > @pre_time
    AND visit_start_date > observation_period_start_date
    AND DATEDIFF(DAY, visit_start_date, observation_period_end_date) > @post_time
    AND visit_start_date < observation_period_end_date
  AND DATEDIFF(DAY, condition_start_date, visit_start_date) > @pre_time
  AND condition_start_date <= visit_start_date
  AND condition_concept_id IN (
    SELECT descendant_concept_id
    FROM @cdmDatabaseSchema.concept_ancestor
    WHERE ancestor_concept_id = 201826) /* Type 2 DM */
GROUP BY visit_occurrence.person_id;

INSERT INTO @resultsDatabaseSchema.rehospitalization
SELECT visit_occurrence.person_id AS subject_id,
    visit_start_date AS cohort_start_date,
    visit_end_date AS cohort_end_date,
    2 AS cohort_definition_id
FROM @resultsDatabaseSchema.rehospitalization
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON visit_occurrence.person_id = rehospitalization.subject_id
WHERE visit_concept_id IN (9201, 9203)
    AND visit_start_date > cohort_start_date
    AND visit_start_date <= cohort_end_date
    AND cohort_definition_id = 1;
```

This is parameterized SQL which can be used by the `SqlRender` package. We use parameterized SQL so we do not have to pre-specify the names of the CDM and result schemas. That way, if we want to run the SQL on a different schema, we only need to change the parameter values; we do not have to change the SQL code. By also making use of translation functionality in `SqlRender`, we can make sure the SQL code can be run in many different environments.

```
library(SqlRender)
sql <- readSql("HospitalizationCohorts.sql")
sql <- renderSql(sql,
                 cdmDatabaseSchema = cdmDatabaseSchema,
                 cohortsDatabaseSchema = cohortsDatabaseSchema,
                 post_time = 30,
                 pre_time = 365)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql
```

```
connection <- connect(connectionDetails)
executeSql(connection, sql)
```

In this code, we first read the SQL from the file into memory. In the next line, we replace four parameter names with the actual values. We then translate the SQL into the dialect appropriate for the DBMS we already specified in the `connectionDetails`. Next, we connect to the server, and submit the rendered and translated SQL.

If all went well, we now have a table with the events of interest. We can see how many events per type:

```
sql <- paste("SELECT cohort_definition_id, COUNT(*) AS count",
             "FROM @cohortsDatabaseSchema.rehospitalization",
             "GROUP BY cohort_definition_id")
sql <- renderSql(sql, cohortsDatabaseSchema = cohortsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
##   cohort_definition_id  count
## 1                    1 527616
## 2                    2 221555
```

3.3 Extracting the data from the server

Now we can tell `PatientLevelPrediction` to extract all necessary data for our analysis. This is done using the `FeatureExtractionPackage` available at https://github.com/OHDSI/FeatureExtration. In short the FeatureExtractionPackage allows you to specify which features (covariates) need to be extracted, e.g. all conditions and drug exposures. It also supports the creation of custom covariates. For more detailed information on the FeatureExtraction package see its vignettes.

```
covariateSettings <- createCovariateSettings(useCovariateDemographics = TRUE,
                                             useCovariateConditionOccurrence = TRUE,
                                             useCovariateConditionOccurrenceLongTerm = TRUE,
                                             useCovariateConditionOccurrenceShortTerm = TRUE,
                                             useCovariateConditionOccurrenceMediumTerm = TRUE,
                                             useCovariateConditionEra = TRUE,
                                             useCovariateConditionEraEver = TRUE,
                                             useCovariateConditionEraOverlap = TRUE,
                                             useCovariateConditionGroup = TRUE,
                                             useCovariateDrugExposure = TRUE,
                                             useCovariateDrugExposureLongTerm = TRUE,
                                             useCovariateDrugExposureShortTerm = TRUE,
                                             useCovariateDrugEra = TRUE,
                                             useCovariateDrugEraLongTerm = TRUE,
                                             useCovariateDrugEraShortTerm = TRUE,
                                             useCovariateDrugEraOverlap = TRUE,
                                             useCovariateDrugEraEver = TRUE,
                                             useCovariateDrugGroup = TRUE,
                                             useCovariateProcedureOccurrence = TRUE,
                                             useCovariateProcedureOccurrenceLongTerm = TRUE,
                                             useCovariateProcedureOccurrenceShortTerm = TRUE,
                                             useCovariateProcedureGroup = TRUE,
                                             useCovariateObservation = TRUE,
                                             useCovariateObservationLongTerm = TRUE,
```

```
                                                    useCovariateObservationShortTerm = TRUE,
                                                    useCovariateObservationCountLongTerm = TRUE,
                                                    useCovariateMeasurement = TRUE,
                                                    useCovariateMeasurementLongTerm = TRUE,
                                                    useCovariateMeasurementShortTerm = TRUE,
                                                    useCovariateMeasurementCountLongTerm = TRUE,
                                                    useCovariateMeasurementBelow = TRUE,
                                                    useCovariateMeasurementAbove = TRUE,
                                                    useCovariateConceptCounts = TRUE,
                                                    useCovariateRiskScores = TRUE,
                                                    useCovariateRiskScoresCharlson = TRUE,
                                                    useCovariateRiskScoresDCSI = TRUE,
                                                    useCovariateRiskScoresCHADS2 = TRUE,
                                                    useCovariateRiskScoresCHADS2VASc = TRUE,
                                                    useCovariateInteractionYear = FALSE,
                                                    useCovariateInteractionMonth = FALSE,
                                                    excludedCovariateConceptIds = c(),
                                                    deleteCovariatesSmallCount = 100)
```

The final step for extracting the data is to run the `getPlpData` function and input the connection details, the database schema where the cohorts are stored, the cohort definition ids for the cohort and outcome, and the washoutPeriod which 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.

```
plpData <- getPlpData(connectionDetails = connectionDetails,
                      cdmDatabaseSchema = cdmDatabaseSchema,
                      oracleTempSchema = oracleTempSchema,
                      cohortDatabaseSchema = cohortsDatabaseSchema,
                      cohortTable = "rehospitalization",
                      cohortId = 1,
                      washoutPeriod = 183,
                      covariateSettings = covariateSettings,
                      outcomeDatabaseSchema = cohortsDatabaseSchema,
                      outcomeTable = "rehospitalization",
                      outcomeIds = 2,
                      cdmVersion = cdmVersion)
```

Note that if the cohorts are created in ATLAS the corresponding database schemas need to be selected. There are many additional parameters for the `getPlpData` function which are all documented in the `PatientLevelPrediction` manual. The resulting `plpData` object uses the package `ff` to store information in a way that ensures R does not run out of memory, even when the data are large.

We can get some overall statistics using the generic `summary()` method:

```
summary(plpData)
```

3.4 Saving the data to file

Creating the `plpData` object can take considerable computing time, and it is probably a good idea to save it for future sessions. Because `plpData` uses `ff`, we cannot use R's regular save function. Instead, we'll have to use the `savePlpData()` function:

```
savePlpData(plpData, "rehosp_plp_data")
```

We can use the `loadPlpData()` function to load the data in a future session.

4 Applying additional inclusion criteria

To completely define the prediction problem the final study population is obtained by applying additional constraints on the two earlier defined cohorts, e.g., a minumim time at risk can be enforced (`requireTimeAtRisk, minTimeAtRisk`). In this step it is also possible to redefine the risk window based on the at-risk cohort. For example, if we like the risk window to start 30 days after the at-risk cohort start and end a year later we can set `riskWindowStart = 30` and `riskWindowEnd = 365`. In some cases the risk window needs to start at the cohort end date. This can be achieved by setting `addExposureToStart = TRUE` which adds the cohort (exposure) time to the start date.

In the example below a final population is created using an additional constraint on the washout period, removal of patients with prior outcomes in the year before, and a time at risk definition.

```
population <- createStudyPopulation(plpData,
                                     outcomeId = 2,
                                     includeAllOutcomes = TRUE,
                                     firstExposureOnly = TRUE,
                                     washoutPeriod = 365,
                                     removeSubjectsWithPriorOutcome = TRUE,
                                     priorOutcomeLookback = 365,
                                     riskWindowStart = 1,
                                     requireTimeAtRisk = FALSE,
                                     riskWindowEnd = 365)
```

Note that some of these constraints could also already be applied in the cohort creation step, however, the `createStudyPopulation` function allows you do sensitivity analyses more easily on the already extracted plpData from the database.

5 Model Development

5.1 Defining the model settings

Models	Python	Parameters
Logistic regression with regularization	No	var (starting variance)
Gradient boosting machines	No	ntree (number of trees), max depth (max levels in tree), min rows (minimum data points in in node), learning rate
Random forest	Yes	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)
K-nearest neighbors	No	k (number of neighbours),weighted (weight by inverse frequency)
Naive Bayes	Yes	none

The table shows the currently implemented algorithms and their hyper-parameters. Some of these algorithms are calling python.In the settings function of the algorithm the user can specify a list of eligible values for each hyper-parameter. All possible combinations of the hyper-parameters are included in a so-called grid search using cross-validation on the training set. If a user does not specify any value then the default value is used instead.

For example, if we use the following settings for the gradientBoostingMachine: ntrees=c(100,200), max depth=4 the grid search will apply the gradient boosting machine algorithm with ntrees=100 and max depth=4 plus the default settings for other hyper-parameters and ntrees=200 and max depth=4 plus the default settings for other hyper-parameters. The hyper-parameters that lead to the best cross-validation performance will then be chosen for the final model.

```
gbmModel <- setGradientBoostingMachine(ntrees = c(100, 200), max_depth = 4)
lrModel <- setLassoLogisticRegression()
```

5.2 Model training

The `runPlP` function uses the population, `plpData`, and model settings to train and evaluate the model. Because evaluation using the same data on which the model was built can lead to overfitting, one uses a train-test split of the data or cross-validation. This functionaity is build in the `runPlP` function. We can use the testSplit (person/time) and testFraction parameters to split the data in a 75%-25% split and run the patient-level prediction pipeline:

```
lrResults <- runPlp(population,plpData, modelSettings = lrModel, testSplit = 'person',
                    testFraction = 0.25, nfold = 2)
```

Under the hood the package will now use the `Cyclops` package to fit a large-scale regularized regression using 75% of the data and will evaluate the model on the remaining 25%. A results data structure is returned containing information about the model, its performance etc.

5.3 Saving and loading

You can save and load the model using:

```
savePlpModel(lrResults$model, dirPath = file.path(getwd(), "model"))
plpModel <- loadPlpModel(getwd(), "model")
```

You can save and load the full results structure using:

```
savePlpResult(lrResults, location = file.path(getwd(), "lr"))
lrResults <- loadPlpResult(file.path(getwd(), "lr"))
```

6 Model Evaluation

The runPlp() function returns the trained model and the evaluation of the model on the train/test sets. To generate all the plots run the following code:

```
plotPlp(lrResults, dirPath = getwd())
```

To run individual plots you can use the following functions:

```
testResults <- lrResult$performanceEvaluationTest

plotSparseRoc(testResults, "sparseROC.pdf")

plotSparseCalibration(testResults, "sparseCalibration.pdf")

plotPreferencePDF(testResults, "preferencePDF.pdf")

plotPredictionDistribution(testResults, "predictionDistribution.pdf")

plotGeneralizability(testResults, "generalizability.pdf")

plotVariableScatterplot(testResults, "variableScatterplot.pdf")

plotPrecisionRecall(testResults, "precisionRecall.pdf")

plotDemographicSummary(testResults, "demographicSummary.pdf")
```

These plots are described in more detail in the following paragraphs.

6.1 ROC plot

The ROC plot plots the sensitivity against 1-specificity on the test set. The plot shows how well the model is able to discriminate between the people with the outcome and those without. The dashed diagonal line is the performance of a model that randomly assigns predictions. The higher the area under the ROC plot the better the discrimination of the model.

6.2 Calibration plot

The calibration plot shows how close the predicted risk is to the observed risk. The diagonal dashed line thus indicates a perfectly calibrated model. The ten (or fewer) dots represent the mean predicted values for each quantile plotted against the observed fraction of people in that quantile who had the outcome (observed fraction). The straight black line is the linear regression using these 10 plotted quantile mean predicted vs observed fraction points. The two blue straight lines represented the 95% lower and upper confidence intervals of the slope of the fitted line.

6.3 Preference distribution plots

The preference distribution plots are the preference score distributions corresponding to i) people in the test set with the outcome (red) and ii) people in the test set without the outcome (blue).

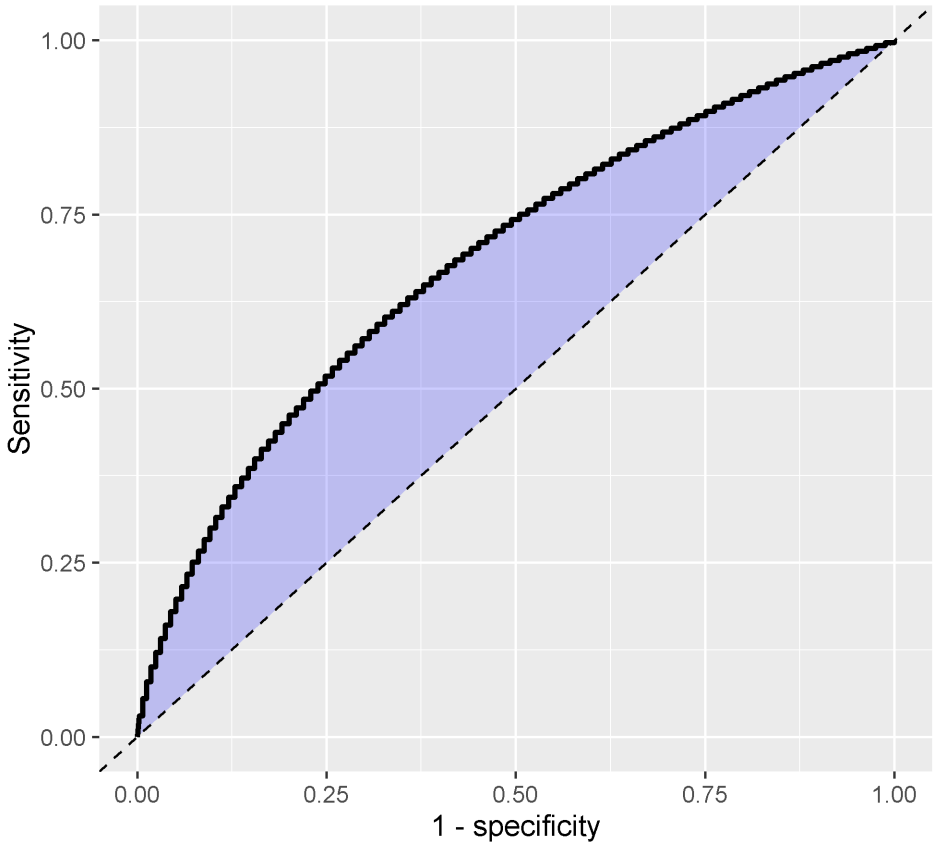


Figure 3: Receiver Operator Plot

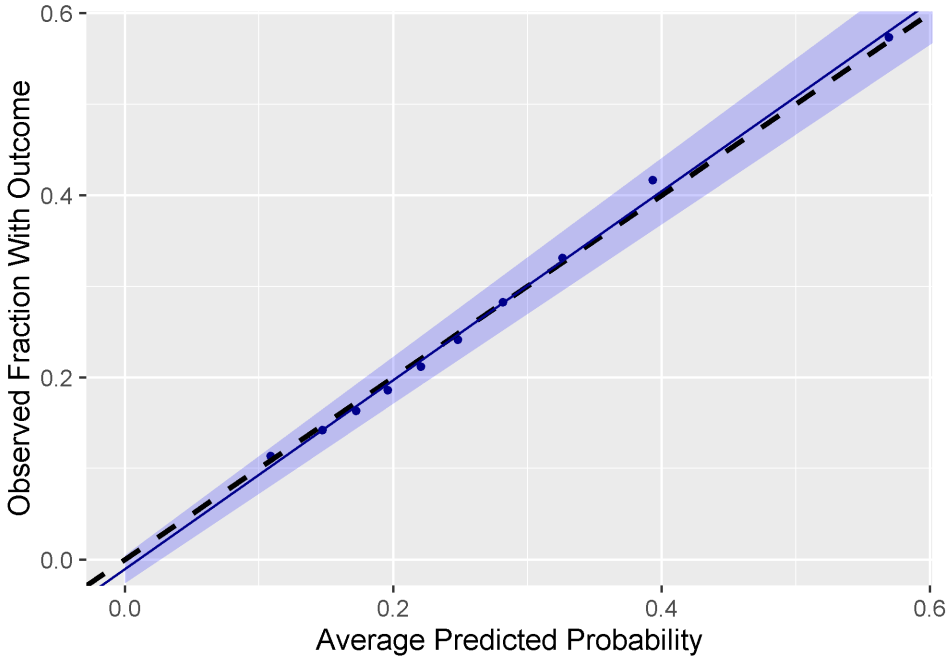


Figure 4: Calibration Plot

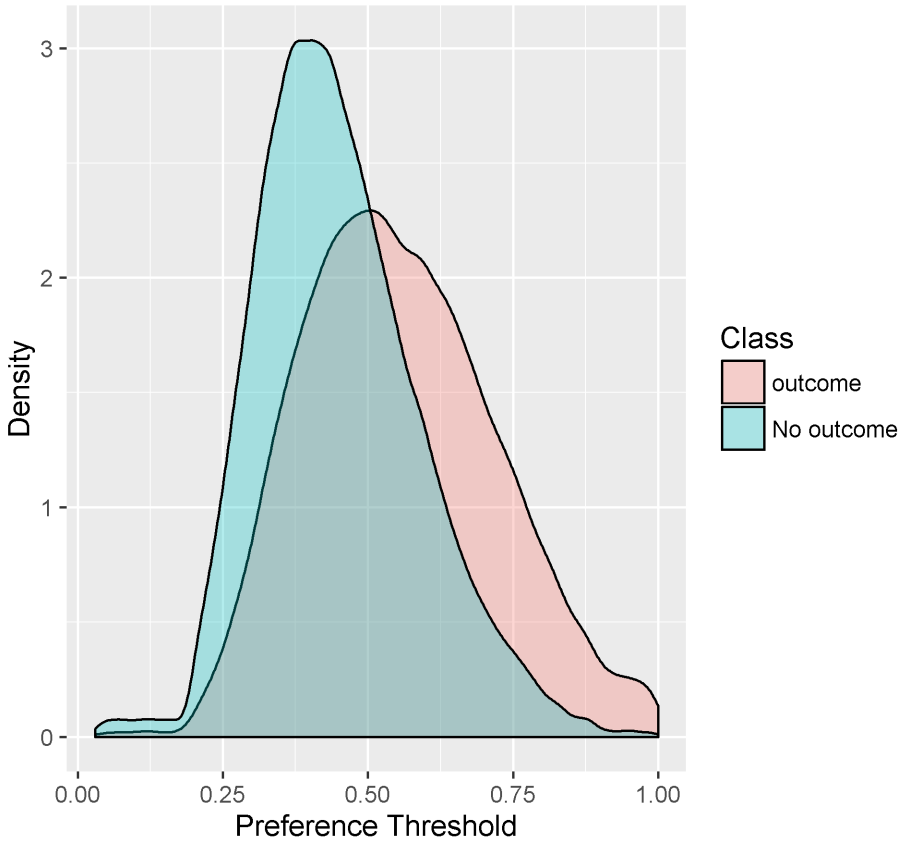


Figure 5: Preference Plot

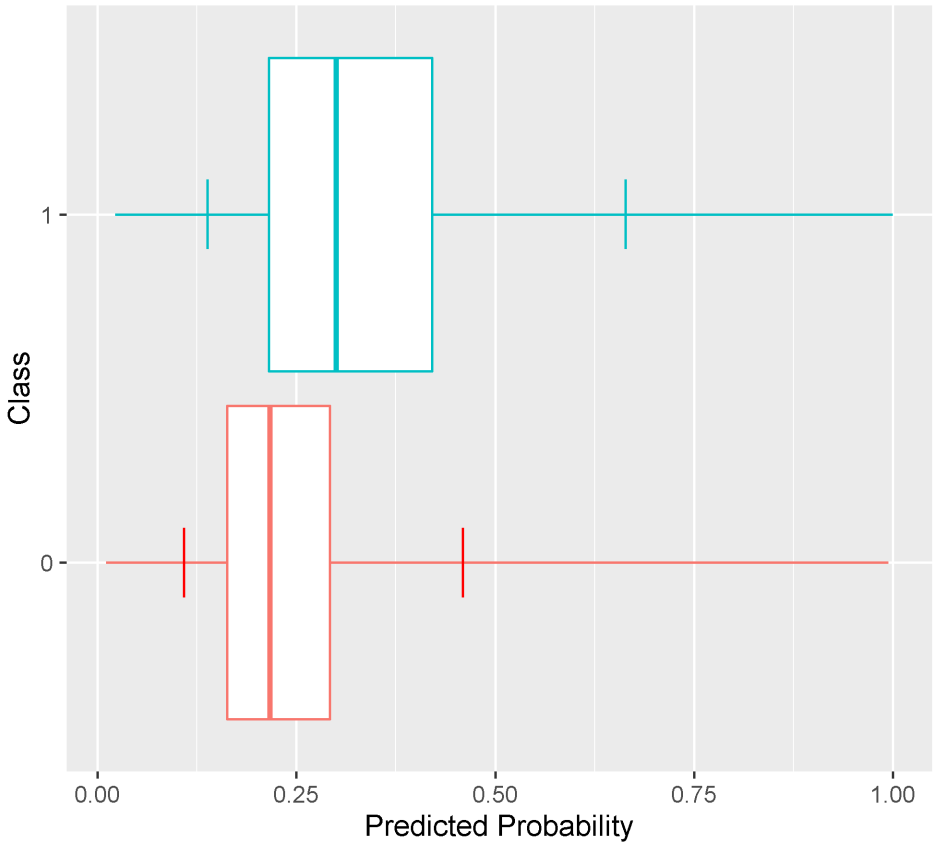


Figure 6: Prediction Distribution Box Plot

6.4 Box plots

The prediction distribution boxplots are box plots for the predicted risks of the people in the test set with the outcome (class 1: blue) and without the outcome (class 0: red).

The box plots in the Figure above show that the predicted probability of the outcome is indeed higher for those with the outcome but there is also overlap between the two distribution which lead to an imperfect discrimination.

6.5 Test-Train similarity plot

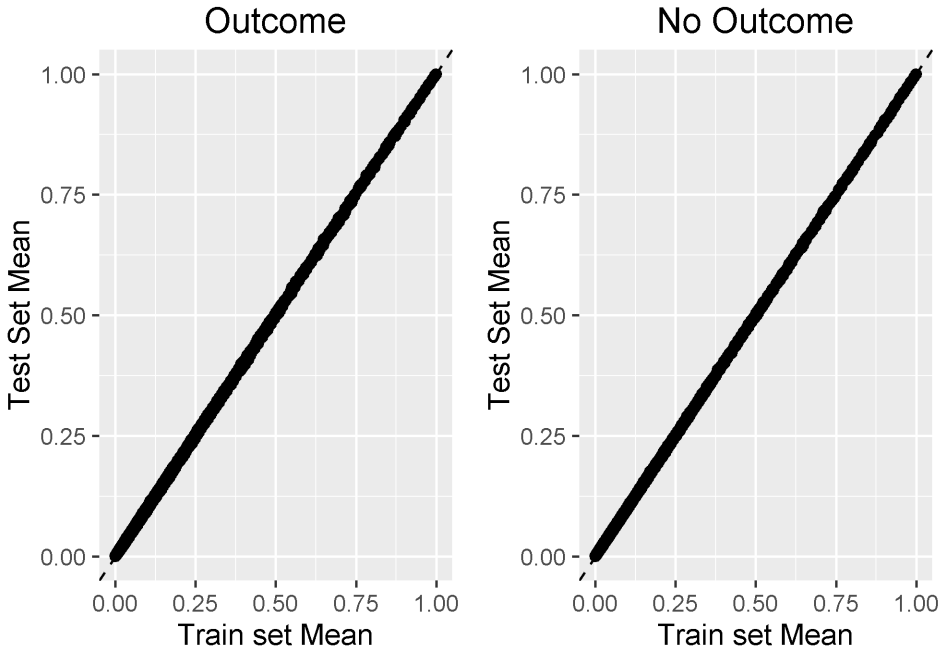


Figure 7: Similarity plots of train and test set

The test-train similarity is presented by plotting the mean covariate values in the train set against those in the test set for people with and without the outcome.

The results for our example of re-hospitalization look very promising since the mean values of the covariates are on the diagonal.

6.6 Variable scatter plot

The variable scatter plot shows the mean covariate value for the people with the outcome against the mean covariate value for the people without the outcome. The size and color of the dots correspond to the importance of the covariates in the trained model (size of beta) and its direction (sign of beta with green meaning positive and red meaning negative), respectivily.

The plot shows that the mean of most of the covariates is higher for subjects with the outcome compared to those without. Also there seem to be a very predictive, but rare covariate with a high beta.

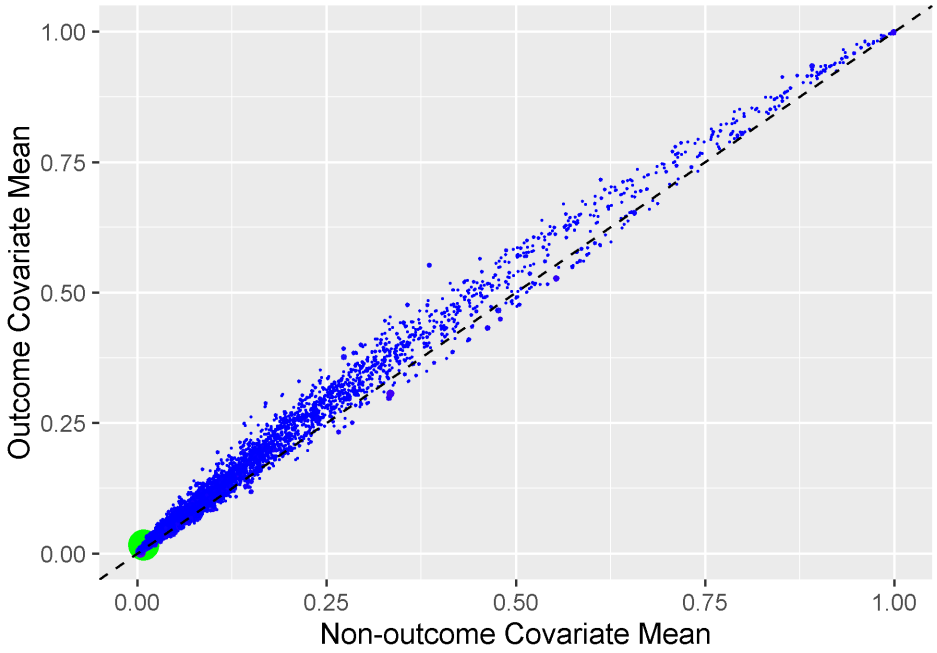
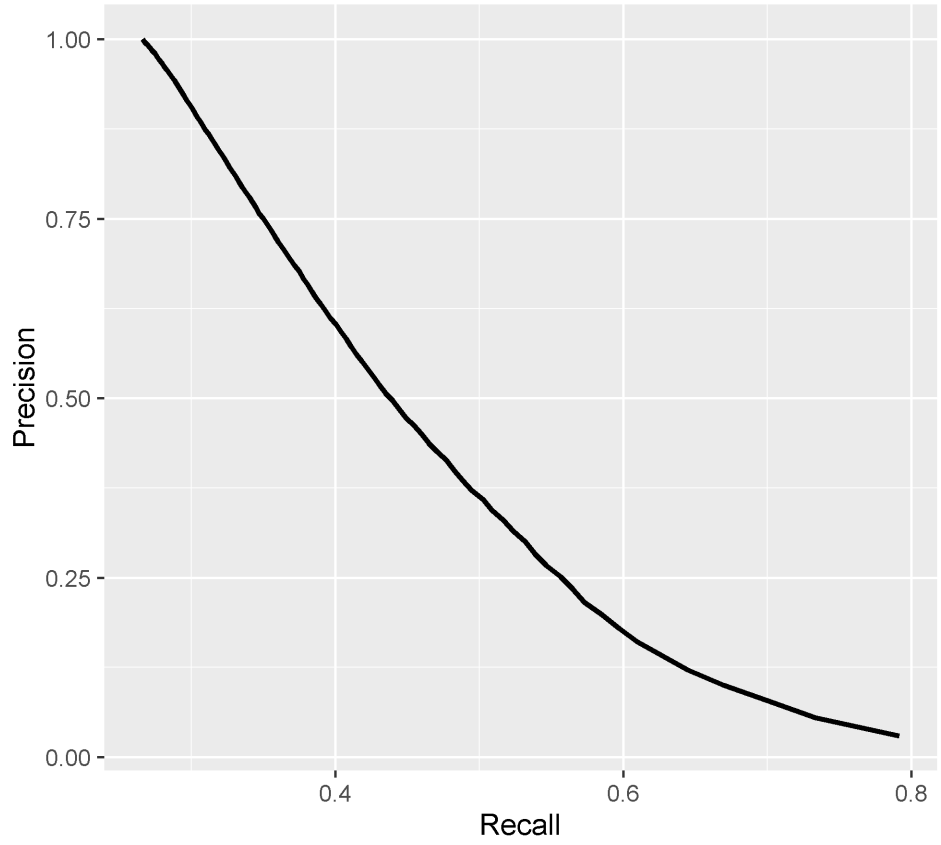


Figure 8: Variabel scatter Plot

6.7 Plot Precision Recall



Precision (P) is defined as the number of true positives (Tp) over the number of true positives plus the number of false positives (Fp).

```
P <- Tp/(Tp + Fp)
```

Recall (R) is defined as the number of true positives (Tp) over the number of true positives plus the number of false negatives (Fn).

```
R <- Tp/(Tp + Fn)
```

These quantities are also related to the (F1) score, which is defined as the harmonic mean of precision and recall.

```
F1 <- 2 * P * R/(P + R)
```

Note that the precision can either decrease or increase if the threshold is lowered. Lowering the threshold of a classifier may increase the denominator, by increasing the number of results returned. If the threshold was previously set too high, the new results may all be true positives, which will increase precision. If the previous threshold was about right or too low, further lowering the threshold will introduce false positives, decreasing precision.

For Recall the demoninator does not depend on the classifier threshold (Tp+Fn is a constant). This means that lowering the classifier threshold may increase recall, by increasing the number of true positive results. It is also possible that lowering the threshold may leave recall unchanged, while the precision fluctuates.

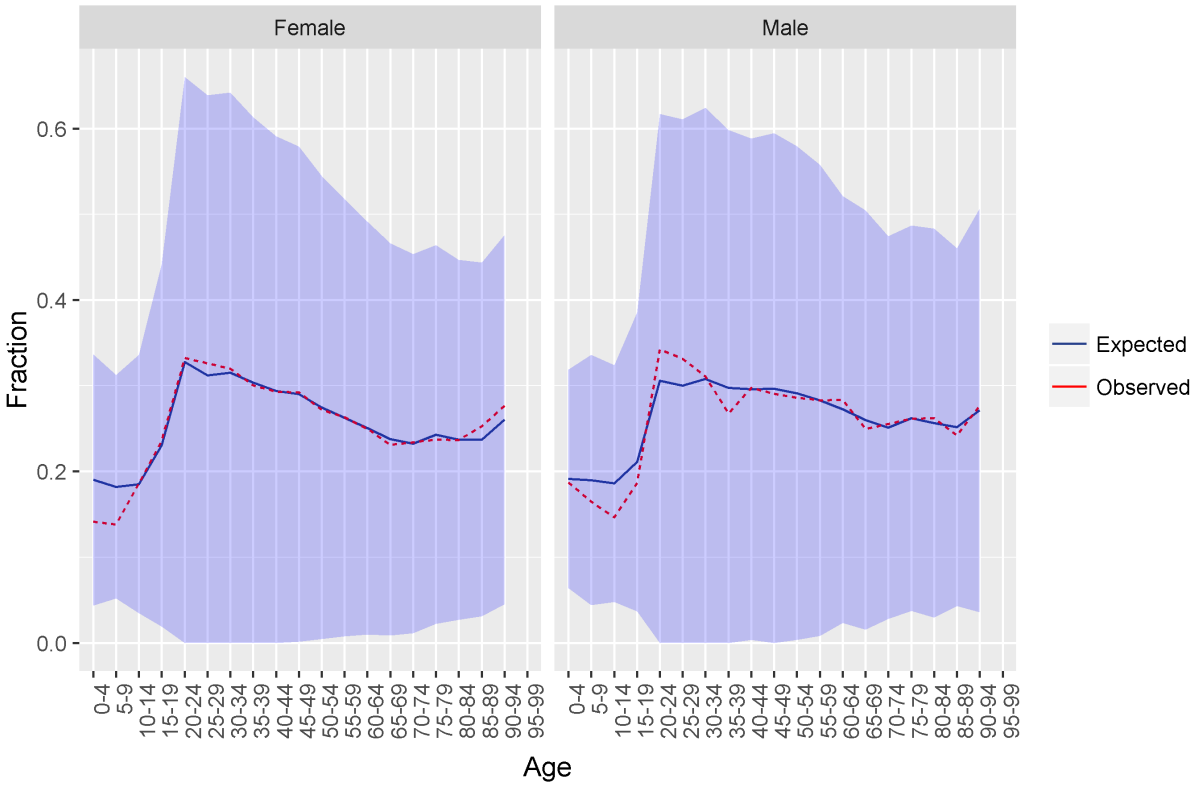


Figure 9: Demographic Summary Plot

6.8 Demographic Summary plot

This plot shows for females and males the expected and observed risk in different age groups together with a confidence area.

7 External validation

We recommend to always peform external validation, i.e. apply the final model on as much new datasets as feasible and evaluate its performance.

```
# load the trained model
plpModel <- loadPlpModel(getwd(), "model")

# load the new plpData and create the population
plpData <- loadPlpData(getwd(), "data")
population <- createStudyPopulation(plpData, outcomeId = 2, includeAllOutcomes = TRUE,
    firstExposureOnly = TRUE, washoutPeriod = 365, removeSubjectsWithPriorOutcome = TRUE,
    priorOutcomeLookback = 365, riskWindowStart = 1, requireTimeAtRisk = FALSE,
    riskWindowEnd = 365)

# apply the trained model on the new data
validationResults <- applyModel(population, plpData, plpModel)
```

8 Acknowledgments

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

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

```
##
##   Jenna Reps, Martijn J. Schuemie, Marc A. Suchard, Patrick B.
##   Ryan and Peter R. Rijnbeek (2017). PatientLevelPrediction:
##   Package for patient level prediction using data in the OMOP
##   Common Data Model. R package version 1.2.1.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {PatientLevelPrediction: Package for patient level prediction using data in the OMOP Comm
## Model},
##     author = {Jenna Reps and Martijn J. Schuemie and Marc A. Suchard and Patrick B. Ryan and Peter R
##     year = {2017},
##     note = {R package version 1.2.1},
##   }
```

Further, `PatientLevelPrediction` makes extensive use of the `Cyclops` package.

```
citation("Cyclops")
```

```
##
## To cite Cyclops in publications use:
##
## Suchard MA, Simpson SE, Zorych I, Ryan P and Madigan D (2013).
## "Massive parallelization of serial inference algorithms for
## complex generalized linear models." _ACM Transactions on Modeling
## and Computer Simulation_, *23*, pp. 10. <URL:
## http://dl.acm.org/citation.cfm?id=2414791>.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {M. A. Suchard and S. E. Simpson and I. Zorych and P. Ryan and D. Madigan},
##     title = {Massive parallelization of serial inference algorithms for complex generalized linear m
##     journal = {ACM Transactions on Modeling and Computer Simulation},
##     volume = {23},
##     pages = {10},
##     year = {2013},
##     url = {http://dl.acm.org/citation.cfm?id=2414791},
##   }
```

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