Building patient-level predictive models

Martijn J. Schuemie, Marc A. Suchard and Patrick Ryan

2015-03-31

Contents

1 Introduction

This vignette describes how you can use the `PatientLevelPrediction` package to build patient-level prediction models. We will walk through all the steps needed to build an exemplar model, and we have selected the well-studied topic of predicting re-hospitalization.

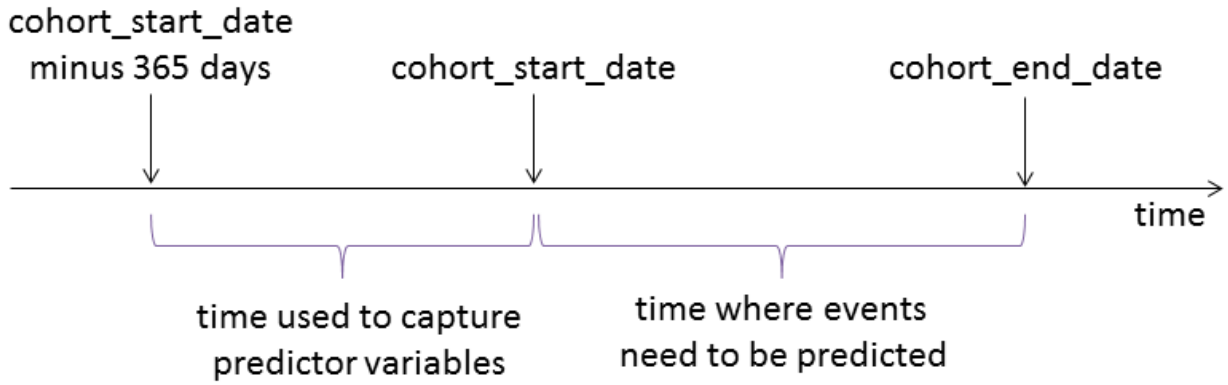
1.1 Specifying the cohort of interest and outcomes

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 cohort and outcomes should be provided as data in a table on the server, where the table should have the same structure as the cohort table in the OMOP CDM, meaning it should have the following columns:

- `cohort_concept_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`, the end of the time period. Can be NULL for outcomes.



The package will use data from the time period preceding (and including) the `cohort_start_date` to build a large set of features that can be used to predict outcomes between (and including) the `cohort_start_date` and `cohort_end_date`. These features can include binary indicators for the occurrence of any individual drug, condition, procedures, as well as demographics and comorbidity indices.

Important: Currently, `PatientLevelPrediction` does not check or limit the input data in any way. It is up to the user to make sure there are at least 365 days of observation time preceding the `cohort_start_date` for the cohort of interest. Furthermore, since the `cohort_start_date` is included in both the time used to construct predictors, and the time when outcomes can occurr, it is up to the user to either remove outcomes occurring on the `cohort_start_date`, or to remove predictors that are in fact indicators of the occurrence of an outcome on the `cohort_start_date`.

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.

The `PatientLevelPrediction` package is currently maintained in a Github repository, and has dependencies on other packages in Github. All of these packages can be downloaded and installed from within R using the `devtools` package:

```
install.packages("devtools")
library(devtools)
install_github("ohdsi/SqlRender")
install_github("ohdsi/DatabaseConnector")
install_github("ohdsi/Cyclops")
install_github("ohdsi/PatientLevelPrediction")
```

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

3 Data extraction

The first step in running the PatientLevelPrediction is extracting all necessary data from the database server holding the data in the Common Data Model (CDM) format.

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"
resultsDatabaseSchema <- "my_results"
```

The last two lines define the cdmDatabaseSchema and resultsDatabaseSchema variables, which we'll use later to tell R where the data in CDM format live, and where we want to write intermediate and result tables. Note that for Microsoft SQL Server, these variables 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

Before we can start using the PatientLevelPrediction package itself, we need to construct a cohort of interest for which we want to perform the prediction, and the outcome, the event that we would like to predict.We do this by writing SQL statements against the CDM that populate a table containing the persons and events of interest. For our example study, we need to create the cohort of persons that have been hospitalized and have a minimum amount of observation time available before and after the hospitalization. We also need to defined 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_concept_id
INTO @resultsDatabaseSchema.rehospitalization
FROM @cdmDatabaseSchema.visit_occurrence
INNER JOIN @cdmDatabaseSchema.observation_period
    ON visit_occurrence.person_id = observation_period.person_id
WHERE place_of_service_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
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_concept_id
FROM @resultsDatabaseSchema.rehospitalization
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON visit_occurrence.person_id = rehospitalization.subject_id
WHERE place_of_service_concept_id IN (9201, 9203)
    AND visit_start_date > cohort_start_date
    AND visit_start_date <= cohort_end_date
    AND cohort_concept_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,
                 resultsDatabaseSchema = resultsDatabaseSchema,
                 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_concept_id, COUNT(*) AS count",
             "FROM @resultsDatabaseSchema.rehospitalization",
             "GROUP BY cohort_concept_id")
sql <- renderSql(sql, resultsDatabaseSchema = resultsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
##   cohort_concept_id   count
## 1                 1 4416811
## 2                 2 1093943
```

3.3 Extracting the data from the server

Now we can tell `PatientLevelPrediction` to extract all necessary data for our analysis:

```
cohortData <- getDbCohortData(connectionDetails$dbms,
                              cdmDatabaseSchema = cdmDatabaseSchema,
                              cohortDatabaseSchema = resultsDatabaseSchema,
                              cohortTable = "rehospitalization",
                              cohortConceptIds = 1)

covariateSettings <- createCovariateSettings(useCovariateDemographics = TRUE,
                                             useCovariateConditionOccurrence = TRUE,
                                             useCovariateConditionOccurrence365d = TRUE,
                                             useCovariateConditionOccurrence30d = TRUE,
                                             useCovariateConditionOccurrenceInpt180d = TRUE,
                                             useCovariateConditionEra = TRUE,
                                             useCovariateConditionEraEver = TRUE,
                                             useCovariateConditionEraOverlap = TRUE,
                                             useCovariateConditionGroup = TRUE,
                                             useCovariateDrugExposure = TRUE,
                                             useCovariateDrugExposure365d = TRUE,
                                             useCovariateDrugExposure30d = TRUE,
                                             useCovariateDrugEra = TRUE,
                                             useCovariateDrugEra365d = TRUE,
                                             useCovariateDrugEra30d = TRUE,
                                             useCovariateDrugEraOverlap = TRUE,
                                             useCovariateDrugEraEver = TRUE,
                                             useCovariateDrugGroup = TRUE,
                                             useCovariateProcedureOccurrence = TRUE,
                                             useCovariateProcedureOccurrence365d = TRUE,
                                             useCovariateProcedureOccurrence30d = TRUE,
                                             useCovariateProcedureGroup = TRUE,
                                             useCovariateObservation = TRUE,
                                             useCovariateObservation365d = TRUE,
                                             useCovariateObservation30d = TRUE,
                                             useCovariateObservationBelow = TRUE,
                                             useCovariateObservationAbove = TRUE,
                                             useCovariateObservationCount365d = TRUE,
                                             useCovariateConceptCounts = TRUE,
                                             useCovariateRiskScores = TRUE,
                                             useCovariateInteractionYear = FALSE,
                                             useCovariateInteractionMonth = FALSE,
                                             excludedCovariateConceptIds = c(),
                                             deleteCovariatesSmallCount = 1)
covariateData <- getDbCovariateData(connectionDetails$dbms,
                                    cdmDatabaseSchema = cdmDatabaseSchema,
                                    cohortDatabaseSchema = resultsDatabaseSchema,
                                    cohortTable = "rehospitalization",
                                    cohortConceptIds = 1,
```

```
                                    covariateSettings = covariateSettings)

outcomeData <- getDbOutcomeData(connectionDetails$dbms,
                                cdmDatabaseSchema = cdmDatabaseSchema,
                                cohortDatabaseSchema = resultsDatabaseSchema,
                                cohortTable = "rehospitalization",
                                cohortConceptIds = 1,
                                outcomeDatabaseSchema = resultsDatabaseSchema,
                                outcomeTable = "rehospitalization",
                                outcomeConceptIds = 2)
```

There are many parameters, but they are all documented in the `PatientLevelPrediction` manual. In short, we are pointing the functions to the table created earlier and indicating which concept IDs in that table identify the cohort and outcome of interest. We construct `cohortData`, an object representing the cohort of interest, `covariateData`, holding all covariate information that will be used as predictors, and `outcomeData` as the set of outcomes during cohort time. All three objects use 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(cohortData)
```

```
## CohortData object summary
##
##   Cohort count Person count
## 1       136506       136506
```

```
summary(covariateData)
```

```
## CovariateData object summary
##
## Number of covariates: 85309
## Number of non-zero covariate values: 19043472
```

```
summary(outcomeData)
```

```
## OutcomeData object summary
##
##   Cohort count Person count
## 2         3460         3460
```

3.3.1 Saving the data to file

Creating the three data objects can take considerable computing time, and it is probably a good idea to save them for future sessions. Because `cohortData`, `covariateData` and ‘`outcomeData` use `ff`, we cannot use R’s regular save function. Instead, we’ll have to use the `saveCohortData()`, `saveCovariateData()`, and `saveOutcomeData()` functions:

```
saveCohortData(cohortData, "rehosp_cohorts")
saveCovariateData(covariateData, "rehosp_covariates")
saveOutcomeData(outcomeData, "rehos_outcomes")
```

We can use the `loadCohortData()`, `loadCovariateData()`, and `loadOutcomeData()` function to load the data in a future session.

4 Fitting the model

4.1 Train-test split

We typically not only want to build our model, we also want to know how good it is. Because evaluation using the same data on which the model was fitted can lead to overestimation, one uses a train-test split of the data or cross-validation. We can use the `splitData()` function to split the data in a 75%-25% split:

```
parts <- splitData(cohortData, covariateData, outcomeData, c(0.75, 0.25))
```

The `parts` variable is a list, where each item holds the data for a subset of the full cohort of interest. For example, `parts[[1]]$cohortData` contains the cohort data for 75% of the people in the cohort of interest. We can now fit the model on the first part of the data (the training set):

4.2 Fitting the model on the training data

```
model <- fitPredictiveModel(parts[[1]]$cohortData, parts[[1]]$covariateData,
    parts[[1]]$outcomeData, modelType = "logistic")
```

The `fitPredictiveModel()` function uses the `Cyclops` package to fit a large-scale regularized regression. To fit the model, `Cyclops` needs to know the hyperparameter value which specifies the variance of the prior. By default `Cyclops` will use cross-validation to estimate the optimal hyperparameter. However, be aware that this can take a really long time. You can use the `prior` and `control` parameters of the `fitPredictiveModel()` to specify `Cyclops` behaviour, including using multiple CPUs to speed-up the cross-validation.

4.3 Model evaluation

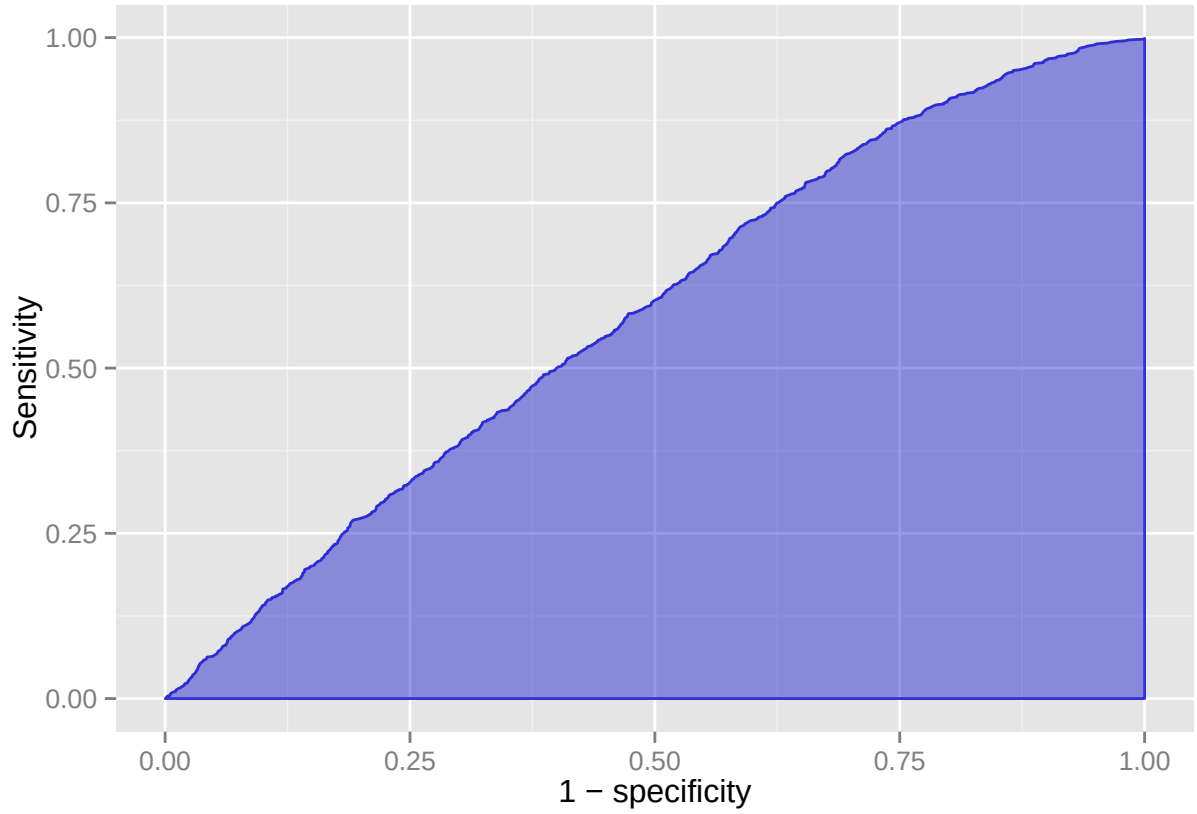
We can evaluate how well the model is able to predict the outcome, for example on the test data we can compute the area under the ROC curve:

```
prediction <- predictProbabilities(model, parts[[2]]$cohortData, parts[[2]]$covariateData)
computeAuc(prediction, parts[[2]]$outcomeData)
```

```
## [1] 0.5811612
```

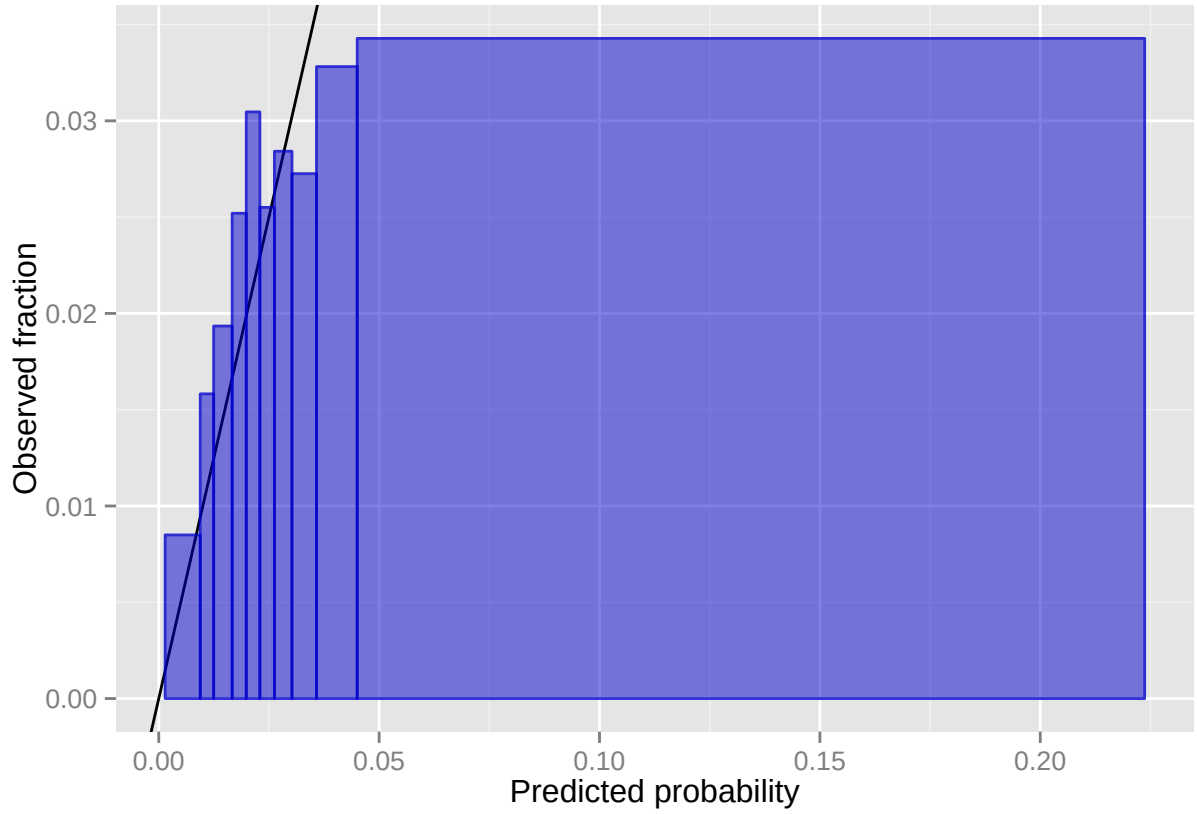
And we can plot the ROC curve itself:

```
plotRoc(prediction, parts[[2]]$outcomeData)
```



We can also look at the calibration:

```
plotCalibration(prediction, parts[[2]]$outcomeData, numberOfStrata = 10)
```



4.4 Inspecting the model

Now that we have some idea of the operating characteristics of our predictive model, we can investigate the model itself:

```
modelDetails <- getModelDetails(model, covariateData)
head(modelDetails)
```

```
##            coefficient         id
## 0           -2.0892606          0
## 2002        -1.5712713       2002
## 2003        -1.1606206       2003
## 2004        -0.8796381       2004
## 1149380504   0.8707636 1149380504
## 1           -0.8702121          1
##                                        covariateName
## 0                                          Intercept
## 2002                                Index year: 2002
## 2003                                Index year: 2003
## 2004                                Index year: 2004
## 1149380504 ...or to cohort index:  1149380-fluticasone
## 1                                Cohort definition ID
```

This shows the strongest predictors in the model with their corresponding betas.

5 Acknowledgments

Considerable work has been dedicated to provide the PatientLevelPrediction package.

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

```
##
## To cite package 'PatientLevelPrediction' in publications use:
##
##   Martijn J. Schuemie, Marc A. Suchard and Patrick B. Ryan (2015).
##   PatientLevelPrediction: Package for patient level prediction
##   using data in the OMOP Common Data Model. R package version
##   0.0.1.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {PatientLevelPrediction: Package for patient level prediction using data in the OMOP Comm
##     author = {Martijn J. Schuemie and Marc A. Suchard and Patrick B. Ryan},
##     year = {2015},
##     note = {R package version 0.0.1},
##   }
##
## ATTENTION: This citation information has been auto-generated from
## the package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
```

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.