07_3_Machine_Learning_Models

Yue Xiong, LMU, yue.xiong@stat.uni-muenchen.de

Machine Learning – Model Training and Evaluation

Introduction

In this tutorial, we'll discuss how to formulate a policy problem or a social science question in the machine learning framework; how to transform raw data into something that can be fed into a model; how to build, evaluate, compare, and select models; and how to reasonably and accurately interpret model results. You'll also get hands-on experience using the `mlr3` package in R.

This tutorial is based on chapter "Machine Learning" of Big Data and Social Science.

Setup

```
library('dbplyr')  # load the corresponding libraries
library('dplyr')
library('RSQLite')
library('glue')
library('caret')
```

```
# Establish a connection to the ncdoc.db database
database_path = "F:/hiwi_work_notebook/ncdoc.db"
conn = DBI::dbConnect(SQLite(), database_path)
src_dbi(conn)
```

```
## src:  sqlite 3.36.0 [F:\hiwi_work_notebook\ncdoc.db]
## tbls: feature_age_2008, feature_age_2013, feature_age_first_admit,
##   feature_agefirstadmit, feature_length_sentence_2000_2008,
##   feature_length_sentence_2000_2013, feature_num_admits_2000_2008,
##   feature_num_admits_2000_2013, features_2000_2008, features_2000_2013, inmate,
##   offender, recidivism_labels_2009_2013, recidivism_labels_2014_2018,
##   sentences, sentences_prep, test_matrix, train_matrix
```

Problem Formulation

Our Machine Learning Problem Of all prisoners released, we would like to predict who is likely to reenter jail within *5* years of the day we make our prediction. For instance, say it is Jan 1, 2009 and we want to identify which prisoners are likely to re-enter jail between now and end of 2013. We can run our predictive model and identify who is most likely at risk. The is an example of a *binary classification* problem.

Note the outcome window of 5 years is completely arbitrary. You could use a window of 5, 3, 1 years or 1 day.

In order to predict recidivism, we will be using data from the `inmate` and `sentences` table to create labels (predictors, or independent variables, or X variables) and features (dependent variables, or Y variables).

We need to munge our data into **labels** (1_Machine_Learning_Labels.rmd) and **features** (2_Machine_Learning_Features.rmd) before we can train and evaluate **machine learning models** (3_Machine_Learning_Models.rmd).

This tutorial assumes that you have already worked through the `1_Machine_Learning_Labels` and `2_Machine_Learning_Features` tutorials. In the following chunk, we will recreate the two function `create_labels` and `create_features` again.

```
# create labels function
create_labels <- function(features_end, prediction_start, prediction_end, conn) {
  # @param features_end
  # @param prediction_start
  # @param prediction_end
  # @param conn: obj

  end_x_year = format(as.Date(features_end, format="%Y-%m-%d"),"%Y")
  start_y_year = format(as.Date(prediction_start, format="%Y-%m-%d"),"%Y")
  end_y_year = format(as.Date(prediction_end, format="%Y-%m-%d"),"%Y")

  drop_script_1 = "drop table if exists sentences_prep;"
  sql_script_1 = glue("create table sentences_prep as
select inmate_doc_number, cast(inmate_sentence_component as integer)
as sentence_component, date([sentence_begin_date_(for_max)])
as sentence_begin_date,
date(actual_sentence_end_date) as sentence_end_date
from sentences;")

  drop_script_2 = glue("drop table if exists release_dates_2000_{end_x_year};")
  sql_script_2 = glue("create temp table release_dates_2000_{end_x_year} as
select inmate_doc_number, sentence_end_date
from sentences_prep where sentence_end_date >= '2000-01-01' and
sentence_end_date <= '{features_end}';")

  drop_script_3 = glue("drop table if exists last_exit_2000_{end_x_year};")
  sql_script_3 = glue("create temp table last_exit_2000_{end_x_year} as
select inmate_doc_number, max(sentence_end_date) sentence_end_date
from release_dates_2000_{end_x_year} group by inmate_doc_number;")

  drop_script_4 = glue("drop table if exists admit_{start_y_year}_{end_y_year};")
  sql_script_4 = glue("create temp table admit_{start_y_year}_{end_y_year} as
select inmate_doc_number, sentence_component, sentence_begin_date
from sentences_prep where sentence_begin_date >= '{prediction_start}'
and sentence_begin_date <= '{prediction_end}' and sentence_component = 1;")

  drop_script_5 = glue("drop table if exists recidivism_{start_y_year}_{end_y_year};")
  sql_script_5 = glue("create temp table recidivism_{start_y_year}_{end_y_year} as
select r.inmate_doc_number, r.sentence_end_date, a.sentence_begin_date, case
when a.sentence_begin_date is null then 0 else 1 end recidivism
from last_exit_2000_{end_x_year} r
left join admit_{start_y_year}_{end_y_year} a on r.inmate_doc_number = a.inmate_doc_number;")

  drop_script_6 = glue("drop table if exists recidivism_labels_{start_y_year}_{end_y_year};")
```

```
  sql_script_6 = glue("
create table recidivism_labels_{start_y_year}_{end_y_year} as
select distinct inmate_doc_number, recidivism
from recidivism_{start_y_year}_{end_y_year};")

  DBI::dbSendStatement(conn, drop_script_1)
  DBI::dbSendQuery(conn, sql_script_1)

  DBI::dbSendStatement(conn, drop_script_2)
  DBI::dbSendQuery(conn, sql_script_2)

  DBI::dbSendStatement(conn, drop_script_3)
  DBI::dbSendQuery(conn, sql_script_3)

  DBI::dbSendStatement(conn, drop_script_4)
  DBI::dbSendQuery(conn, sql_script_4)

  DBI::dbSendStatement(conn, drop_script_5)
  DBI::dbSendQuery(conn, sql_script_5)

  DBI::dbSendStatement(conn, drop_script_6)
  DBI::dbSendQuery(conn, sql_script_6)

  sql_query = glue("select * from recidivism_labels_{start_y_year}_{end_y_year}")
  df_label = data.frame(tbl(conn, sql(sql_query)))

  return(df_label)
}
```

```
# create features function
create_features <- function(features_end, prediction_start, prediction_end, conn) {
  # @param features_end
  # @param prediction_start
  # @param prediction_end
  # @param conn: obj
  end_x_year = format(as.Date(features_end, format="%Y-%m-%d"),"%Y")
  start_y_year = format(as.Date(prediction_start, format="%Y-%m-%d"),"%Y")
  end_y_year = format(as.Date(prediction_end, format="%Y-%m-%d"),"%Y")

  drop_script_1 = "drop table if exists sentences_prep;"
  sql_script_1 = glue("create table sentences_prep as
select inmate_doc_number,
cast(inmate_sentence_component as integer) as sentence_component,
date([sentence_begin_date_(for_max)]) as sentence_begin_date,
date(actual_sentence_end_date) as sentence_end_date
from sentences;")

  drop_script_2 = glue("drop table if exists feature_num_admits_2000_{end_x_year};")
  sql_script_2 = glue("create table feature_num_admits_2000_{end_x_year} as
select inmate_doc_number, count(*) num_admits from sentences_prep
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_{start_y_year}_{end_y_year}
and sentence_begin_date < '{features_end}' and sentence_component = 1
group by inmate_doc_number;")
```

```
  drop_script_3 = glue("drop table if exists feature_length_sentence_2000_{end_x_year};")
  sql_script_3 = glue("create table feature_length_sentence_2000_{end_x_year} as
select inmate_doc_number, sentence_component, cast(julianday(sentence_end_date) - julianday(sentence_beg
from sentences_prep
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_{start_y_year}_{end_y_year}
and sentence_begin_date < '{features_end}' and sentence_component = 1
and sentence_begin_date > '0001-01-01' and sentence_end_date > '0001-01-01' and sentence_end_date > sen

  drop_script_4 = glue("drop table if exists feature_length_long_sentence_2000_{end_x_year};")
  sql_script_4 = glue("create temp table feature_length_long_sentence_2000_{end_x_year} as
select inmate_doc_number, max(length_sentence) length_longest_sentence
from feature_length_sentence_2000_{end_x_year}
group by inmate_doc_number;")

  drop_script_5 = "drop table if exists docnbr_admityr;"
  sql_script_5 = "create temp table docnbr_admityr as
select inmate_doc_number, min(sentence_begin_date) min_admityr
from sentences_prep
where sentence_begin_date > '0001-01-01'
group by inmate_doc_number;"

  drop_script_6 = "drop table if exists age_first_admit_birth_year;"
  sql_script_6 = 'create temp table age_first_admit_birth_year as
select da.inmate_doc_number,
cast(strftime("%Y", da.min_admityr) as integer) min_admityr,
cast(strftime("%Y", p.inmate_birth_date) as integer) inmate_birth_date
from docnbr_admityr da
left join inmate p on da.inmate_doc_number = p.inmate_doc_number;'

  drop_script_7 = "drop table if exists feature_age_first_admit;"
  sql_script_7 = "create table feature_age_first_admit as
select inmate_doc_number, (min_admityr - inmate_birth_date) age_first_admit
from age_first_admit_birth_year;"

  drop_script_8 = "drop table if exists feature_agefirstadmit;"
  sql_script_8 = glue("create table feature_agefirstadmit as
select inmate_doc_number, age_first_admit
from feature_age_first_admit
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_{start_y_year}_{end_y_year}

  drop_script_9 = glue("drop table if exists feature_age_{end_x_year};")
  sql_script_9 = glue('create table feature_age_{end_x_year} as
select inmate_doc_number, ({end_x_year} - cast(strftime("%Y", inmate_birth_date) as integer)) age
from inmate
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_{start_y_year}_{end_y_year}

  drop_script_10 = glue("drop table if exists features_2000_{end_x_year};")
  sql_script_10 = glue('create table features_2000_{end_x_year} as
select f1.inmate_doc_number, f1.num_admits, f2.length_longest_sentence, f3.age_first_admit, f4.age
from feature_num_admits_2000_{end_x_year} f1
left join feature_length_long_sentence_2000_{end_x_year} f2 on f1.inmate_doc_number = f2.inmate_doc_numb
left join feature_agefirstadmit f3 on f1.inmate_doc_number = f3.inmate_doc_number
left join feature_age_{end_x_year} f4 on f1.inmate_doc_number = f4.inmate_doc_number;')
```

```
DBI::dbSendStatement(conn, drop_script_1)
DBI::dbSendStatement(conn, sql_script_1)

DBI::dbSendStatement(conn, drop_script_2)
DBI::dbSendStatement(conn, sql_script_2)

DBI::dbSendStatement(conn, drop_script_3)
DBI::dbSendStatement(conn, sql_script_3)

DBI::dbSendStatement(conn, drop_script_4)
DBI::dbSendStatement(conn, sql_script_4)

DBI::dbSendStatement(conn, drop_script_5)
DBI::dbSendStatement(conn, sql_script_5)

DBI::dbSendStatement(conn, drop_script_6)
DBI::dbSendStatement(conn, sql_script_6)

DBI::dbSendStatement(conn, drop_script_7)
DBI::dbSendStatement(conn, sql_script_7)

DBI::dbSendStatement(conn, drop_script_8)
DBI::dbSendStatement(conn, sql_script_8)

DBI::dbSendStatement(conn, drop_script_9)
DBI::dbSendStatement(conn, sql_script_9)

DBI::dbSendStatement(conn, drop_script_10)
DBI::dbSendStatement(conn, sql_script_10)

sql_query = glue("select * from features_2000_{end_x_year}")
df_features = data.frame(tbl(conn, sql(sql_query)))

return(df_features)}
```

Create Training and Test Sets

The Machine Learning framework we will be using is called "mlr3", which provides an Efficient, object-oriented programming platform on the building blocks of machine learning. In order to start training and evaluation with "mlr3", we are required to follow these procedures listed: - Define the **Task**. - Define the **Learner**. - Implement the training. - Implement the prediction. - Evaluate the trained model with multiple **Measures**.

However, the "mlr3" machine learning framework utilize a resampling framework with cross-validation, bootstrap, etc, included, which does not necessarily take in the training and test splitting beforehand. As an outcome, we will modify the initial configurations according to the python notebooks and combine the training and test dataset as a whole so as to facilitate the `mlr3` training framework.

Our Training Set

We create a training set that takes people at the beginning of 2009 and defines the outcome based on data from 2009-2013 (`recidivism_labels_2009_2013`). The features for each person are based on data up to

the end of 2008 (`features_2000_2008`).

Note: It is important to segregate your data based on time when creating features. Otherwise there can be "leakage", where you accidentally use information that you would not have known at the time.

```
sql_string = "drop table if exists train_matrix;"
DBI::dbSendStatement(conn, sql_string)
```

```
## <SQLiteResult>
##   SQL  drop table if exists train_matrix;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string = "
create table train_matrix as
select l.inmate_doc_number, l.recidivism, f.num_admits, f.length_longest_sentence, f.age_first_admit, f
from recidivism_labels_2009_2013 l
left join features_2000_2008 f on f.inmate_doc_number = l.inmate_doc_number;"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table train_matrix as
## select l.inmate_doc_number, l.recidivism, f.num_admits, f.length_longest_sentence, f.age_first_admit
## from recidivism_labels_2009_2013 l
## left join features_2000_2008 f on f.inmate_doc_number = l.inmate_doc_number;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

We then load the training data into `df_training`.

```
sql_string = "SELECT * FROM train_matrix"

df_training = data.frame(tbl(conn, sql(sql_string)))
```

```
## Warning: Closing open result set, pending rows
```

```
head(df_training, n=5)
```

```
##   INMATE_DOC_NUMBER recidivism num_admits length_longest_sentence
## 1           0000028          0          3                    1097
## 2           0000062          0          2                     273
## 3           0000114          0          2                     256
## 4           0000133          0          1                    6209
## 5           0000152          1          1                      76
##   age_first_admit age
## 1              19  36
## 2              36  51
## 3              49  56
## 4              23  39
## 5              37  43
```

Our Test (Validation) Set

In the machine learning process, we want to build models on the training set and evaluate them on the test set. Our test set will use labels from 2014-2018 (`recidivism_labels_2014_2018`), and our features will be based on data up to the end of 2013 (`features_2000_2013`).

```
sql_string = "drop table if exists test_matrix;"
DBI::dbSendStatement(conn, sql_string)
```

```
## <SQLiteResult>
##   SQL  drop table if exists test_matrix;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string = "
create table test_matrix as
select l.inmate_doc_number, l.recidivism, f.num_admits, f.length_longest_sentence, f.age_first_admit, f
from recidivism_labels_2014_2018 l
left join features_2000_2013 f on f.inmate_doc_number = l.inmate_doc_number;"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table test_matrix as
## select l.inmate_doc_number, l.recidivism, f.num_admits, f.length_longest_sentence, f.age_first_admit
## from recidivism_labels_2014_2018 l
## left join features_2000_2013 f on f.inmate_doc_number = l.inmate_doc_number;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

We load the test data into `df_test`.

```
sql_string = "SELECT * FROM test_matrix"

df_test = data.frame(tbl(conn, sql(sql_string)))
```

```
## Warning: Closing open result set, pending rows
```

```
head(df_test, n=5)
```

```
##   INMATE_DOC_NUMBER recidivism num_admits length_longest_sentence
## 1           0000028          0          3                    1097
## 2           0000033          0          5                    5186
## 3           0000035          0          3                    3969
## 4           0000037          0          1                     137
## 5           0000062          0          2                     273
##   age_first_admit age
## 1              19  41
## 2              18  53
## 3              25  47
## 4              34  38
## 5              36  56
```

The next step would be to join the training and test dataset vertically.

```
total_df <- rbind(df_training, df_test)
nrow(total_df)
```

```
## [1] 376893
```

Data Cleaning

Before we proceed to model training, we need to clean our data. First, we check the percentage of missing values.

```
# count the number of missing values row wise
isnan_totaldf_rows = sum(!complete.cases(total_df))

nrows_totaldf = nrow(total_df)

# count the missing value percentage
missing_percent = isnan_totaldf_rows/nrows_totaldf
missing = glue("missing rows ratio:{missing_percent}")
cat(missing)
```

```
## missing rows ratio:0.00953055641786926
```

We see that about 1% of the rows in our data have missing values. In the following, we will drop rows with missing values. Note, however, that better ways for dealing with missing values exist, e.g., general data imputation methods.

```
total_df = na.omit(total_df)
cleaned_df = nrow(total_df) # number of rows after cleaning
```

```
c(nrows_totaldf, cleaned_df) # checking whether the missing data are dropped
```

```
## [1] 376893 373301
```

Let's check if the values of the ages at first admit are reasonable.

```
length(unique(total_df$age_first_admit))
```

```
## [1] 81
```

```
length(total_df$age_first_admit)
```

```
## [1] 373301
```

Looks like this needs some cleaning. We will drop any rows that have age < 14 and > 99.

```
total_df <- total_df[ which(total_df$age_first_admit >= 14
& total_df$age_first_admit <= 99), ]
```

Let's check how much data we still have and how many examples of recidivism are in our training dataset. When it comes to model evaluation, it is good to know what the "baseline" is in our dataset.

```
row_after_cleaning = glue("Number of rows in the dataset: {nrow(total_df)}")
cat(row_after_cleaning)
```

```
## Number of rows in the dataset: 373282
```

```
count(total_df, total_df$recidivism, sort=TRUE)
```

```
##   total_df$recidivism      n
## 1                   0 301704
## 2                   1  71578
```

```
71578/(301704+71578)
```

```
## [1] 0.1917532
```

```
301704+71578
```

```
## [1] 373282
```

We have about 37,0000 examples, and about 20% of those are positive examples (recidivist), which is what we're trying to identify. About 80% of the examples are negative examples (non-recidivist).

Split into features and labels

Here we select our features and outcome variable in a column vector.

```
sel_cols = c('num_admits', 'length_longest_sentence', 'age_first_admit', 'age', 'recidivism')
```

We can now create an X- and y- object to train and evaluate prediction models with `mlr3`.

```
total_df$recidivism = as.factor(total_df$recidivism)
total_data = total_df[sel_cols]
# X_train = df_training[sel_features]
# y_train = df_training[sel_label]
# X_test = df_test[sel_features]
# y_test = df_test[sel_label]
```

```
dim(total_data)
```

```
## [1] 373282      5
```

```
str(total_data)
```

```
## 'data.frame':    373282 obs. of  5 variables:
##  $ num_admits             : int  3 2 2 1 1 2 4 2 1 7 ...
##  $ length_longest_sentence: int  1097 273 256 6209 76 1147 4282 1631 4936 1835 ...
##  $ age_first_admit        : int  19 36 49 23 37 26 22 21 53 19 ...
##  $ age                    : int  36 51 56 39 43 48 56 37 67 41 ...
##  $ recidivism             : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
```

Before the actual training, we can firstly load the required packages.

```
library("mlr3")
library("mlr3learners")
library("mlr3tuning")
```

```
## Loading required package: paradox
```

```
library("mlr3viz")
library("ggplot2")
library("paradox")
library("e1071")
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:mlr3tuning':
##
##     tune
```

```
# library('kknn')
library("ranger")
```

In the next step, we need to define the task as specified in the `mlr3` package.

```
task_df = TaskClassif$new("ml_task", total_data, target = "recidivism")

task_df
```

```
## <TaskClassif:ml_task> (373282 x 5)
## * Target: recidivism
## * Properties: twoclass
## * Features (4):
##   - int (4): age, age_first_admit, length_longest_sentence, num_admits
```

Logistic Regression Also, in order to use the logistic regression model, we are required to initialize the learner.

```
learner_logreg = lrn("classif.log_reg")
print(learner_logreg)
```

```
## <LearnerClassifLogReg:classif.log_reg>
## * Model: -
## * Parameters: list()
## * Packages: stats
## * Predict Type: response
## * Feature types: logical, integer, numeric, character, factor, ordered
## * Properties: loglik, twoclass, weights
```

In order to perform an efficient splitting of data, one could do the following:

```
train_set = sample(task_df$row_ids, 0.8 * task_df$nrow)
test_set = setdiff(task_df$row_ids, train_set)
```

Then, we have 80% of the data for training and the remained 20% for evaluation.

We can train the model on the train_set by specifiying the row_ids:

```
learner_logreg$train(task_df, row_ids = train_set)
```

The fitted model can be accessed via:

```
learner_logreg$model
```

```
##
## Call:  stats::glm(formula = task$formula(), family = "binomial", data = data,
##     model = FALSE)
##
## Coefficients:
##             (Intercept)                      age          age_first_admit
##               -0.3327224                0.0863597               -0.0253902
## length_longest_sentence               num_admits
##                0.0001536               -0.3613044
##
## Degrees of Freedom: 298624 Total (i.e. Null);  298620 Residual
## Null Deviance:       292000
## Residual Deviance: 265400    AIC: 265400
```

```
summary(learner_logreg$model)
```

```
##
## Call:
## stats::glm(formula = task$formula(), family = "binomial", data = data,
##     model = FALSE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3955   0.3063   0.5153   0.6978   4.6621
##
```

```
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -3.327e-01  2.051e-02  -16.23   <2e-16 ***
## age                      8.636e-02  9.115e-04   94.75   <2e-16 ***
## age_first_admit         -2.539e-02  1.096e-03  -23.16   <2e-16 ***
## length_longest_sentence  1.536e-04  6.142e-06   25.01   <2e-16 ***
## num_admits              -3.613e-01  3.549e-03 -101.80   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 291977  on 298624  degrees of freedom
## Residual deviance: 265393  on 298620  degrees of freedom
## AIC: 265403
##
## Number of Fisher Scoring iterations: 10
```

Random Forest Besides the logistic regression algorithm, we can also train the data with random forest, which implements a feature importance algorithm automatically by supplying the additional importance argument `importance="permutation"`.

```
learner_rf = lrn("classif.ranger", importance = "permutation")
learner_rf$train(task_df, row_ids = train_set)
```

```
## Growing trees.. Progress: 13%. Estimated remaining time: 3 minutes, 35 seconds.
## Growing trees.. Progress: 25%. Estimated remaining time: 3 minutes, 6 seconds.
## Growing trees.. Progress: 37%. Estimated remaining time: 2 minutes, 35 seconds.
## Growing trees.. Progress: 50%. Estimated remaining time: 2 minutes, 4 seconds.
## Growing trees.. Progress: 62%. Estimated remaining time: 1 minute, 34 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 1 minute, 5 seconds.
## Growing trees.. Progress: 86%. Estimated remaining time: 35 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 4 seconds.
## Computing permutation importance.. Progress: 41%. Estimated remaining time: 43 seconds.
## Computing permutation importance.. Progress: 83%. Estimated remaining time: 12 seconds.
```

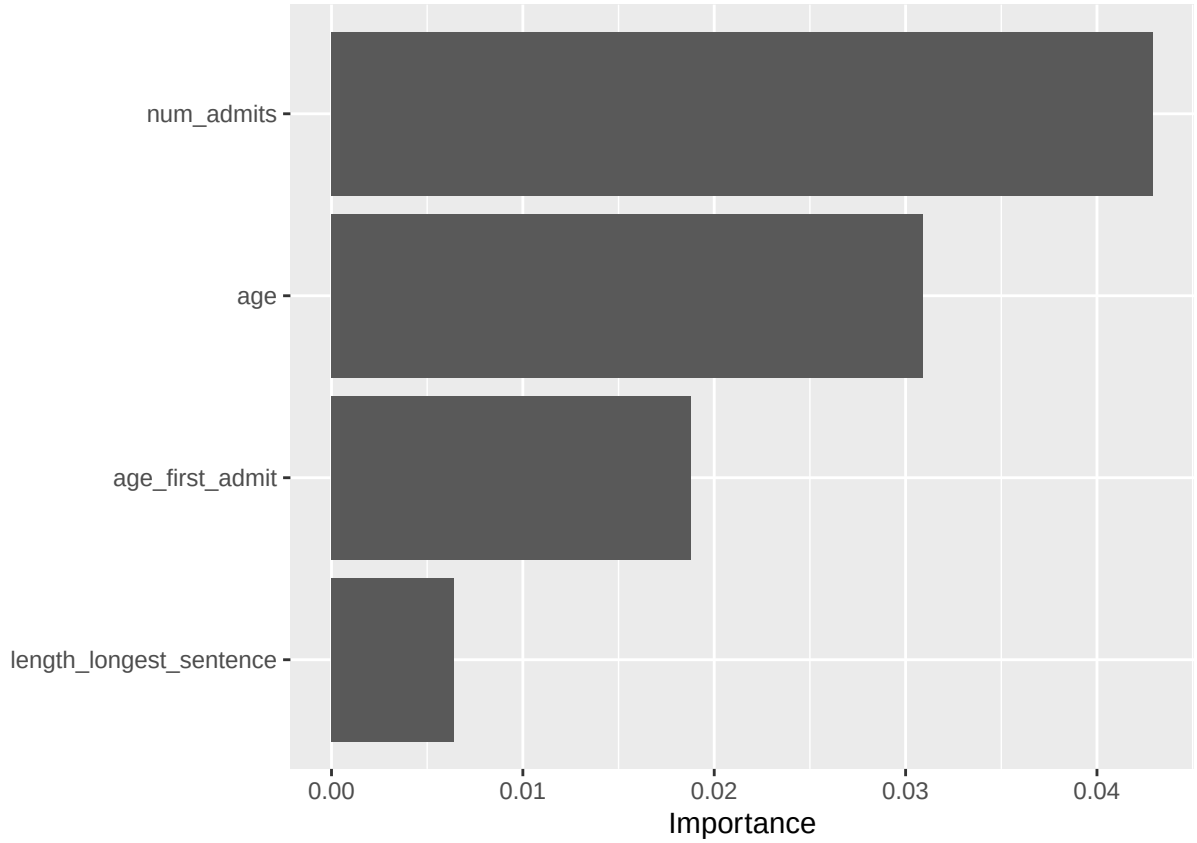
The feature importance values can be seen with `$importance()`:

```
learner_rf$importance()
```

```
##              num_admits                     age         age_first_admit
##             0.042943686             0.030913662             0.018782352
## length_longest_sentence
##             0.006417103
```

We can also plot the importance values as follows:

```
importance = as.data.table(learner_rf$importance(), keep.rownames = TRUE)
colnames(importance) = c("Feature", "Importance")
ggplot(importance, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_col() + coord_flip() + xlab("")
```



Model Prediction

- Predict Class

We can now take a look at the prediction results based on the pre-trained models.

```
pred_logreg = learner_logreg$predict(task_df, row_ids = test_set)
pred_rf = learner_rf$predict(task_df, row_ids = test_set)
```

```
head(as.data.table(pred_logreg), n=5)  # checking the prediction object
```

```
##    row_ids truth response
## 1:       4     0        0
## 2:      14     0        0
## 3:      24     0        0
## 4:      31     0        0
## 5:      49     0        0
```

```
head(as.data.table(pred_rf), n=5)
```

```
##    row_ids truth response
## 1:       4     0        0
## 2:      14     0        0
## 3:      24     0        0
## 4:      31     0        0
## 5:      49     0        0
```

- Predict Probabilities Most learners may not only predict a class variable ("response"), but also their degree of "belief" / "uncertainty" in a given response. Typically, we achieve this by setting the `$predict_type="prob"` in the learner setting arguments.

```
learner_logreg$predict_type = "prob"
```

```
learner_logreg$predict(task_df, row_ids = test_set)
```

```
## <PredictionClassif> for 74657 observations:
##     row_ids truth response    prob.0     prob.1
##           4     0        0 0.9545213 0.04547874
##          14     0        0 0.8072937 0.19270630
##          24     0        0 0.8555983 0.14440169
## ---
##      373269     0        0 0.7148358 0.28516419
##      373275     0        0 0.9449622 0.05503784
##      373278     0        0 0.8177506 0.18224942
```

Performance Evaluation

In order to measure the performance of a learner on test data, we ususally simulate the scenario of these data by splitting them into training and test dataset. In this case, numerous resampling methods can be utilized to repeat the splitting process.

For "mlr3", we can specify the resampling method with the `rsmp()` function:

```
resampling = rsmp("holdout", ratio=2/3)
print(resampling)
```

```
## <ResamplingHoldout> with 1 iterations
## * Instantiated: FALSE
## * Parameters: ratio=0.6667
```

"holdout" is a simple train-test splitting method. The next step is to use the `resample()` function to implement the resampling calculation.

```
res = resample(task_df, learner = learner_logreg, resampling = resampling)
```

```
## INFO  [12:17:26.731] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 1/1)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
res
```

```
## <ResampleResult> of 1 iterations
## * Task: ml_task
## * Learner: classif.log_reg
## * Warnings: 0 in 0 iterations
## * Errors: 0 in 0 iterations
```

The performance score can evaluated using `$aggregate()`:

```
res$aggregate(msrs(c("classif.ce", "classif.auc")))
```

```
##  classif.ce classif.auc
##   0.1927717   0.7163821
```

We can also run cross-validation:

```
resampling = rsmp("cv", folds = 5)
rr = resample(task_df, learner = learner_logreg, resampling = resampling)
```

```
## INFO  [12:17:29.045] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 1/5)
## INFO  [12:17:31.452] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 5/5)
## INFO  [12:17:33.610] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 3/5)
## INFO  [12:17:35.977] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 2/5)
## INFO  [12:17:38.134] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 4/5)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
rr$aggregate(msrs(c("classif.ce", "classif.auc")))
```

```
##  classif.ce classif.auc
##   0.1919139   0.7144090
```

Performance Comparison and Benchmarks

The `benchmark()` function can integrate multiple tasks and learners together. Here, we only show the instance with multiple learners:

```
learners = lrns(c("classif.log_reg", "classif.ranger"), predict_type = "prob")
bm_design = benchmark_grid(
  tasks = task_df,
  learners = learners,
  resamplings = rsmp("cv", folds = 3)
)
bmr = benchmark(bm_design)
```

```
## INFO  [12:17:40.717] [mlr3]  Running benchmark with 6 resampling iterations
## INFO  [12:17:40.722] [mlr3]  Applying learner 'classif.ranger' on task 'ml_task' (iter 2/3)
## Growing trees.. Progress: 20%. Estimated remaining time: 2 minutes, 4 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 1 minute, 33 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 1 minute, 2 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 32 seconds.
```

```
## Growing trees.. Progress: 99%. Estimated remaining time: 0 seconds.
## INFO  [12:21:23.035] [mlr3]  Applying learner 'classif.ranger' on task 'ml_task' (iter 1/3)
## Growing trees.. Progress: 20%. Estimated remaining time: 2 minutes, 4 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 1 minute, 33 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 1 minute, 2 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 31 seconds.
## Growing trees.. Progress: 100%. Estimated remaining time: 0 seconds.
## INFO  [12:25:03.276] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 2/3)
## INFO  [12:25:05.141] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 1/3)
## INFO  [12:25:07.130] [mlr3]  Applying learner 'classif.ranger' on task 'ml_task' (iter 3/3)
## Growing trees.. Progress: 20%. Estimated remaining time: 2 minutes, 2 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 1 minute, 32 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 1 minute, 1 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 32 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 2 seconds.
## INFO  [12:28:50.711] [mlr3]  Applying learner 'classif.log_reg' on task 'ml_task' (iter 3/3)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## INFO  [12:28:52.867] [mlr3]  Finished benchmark
```

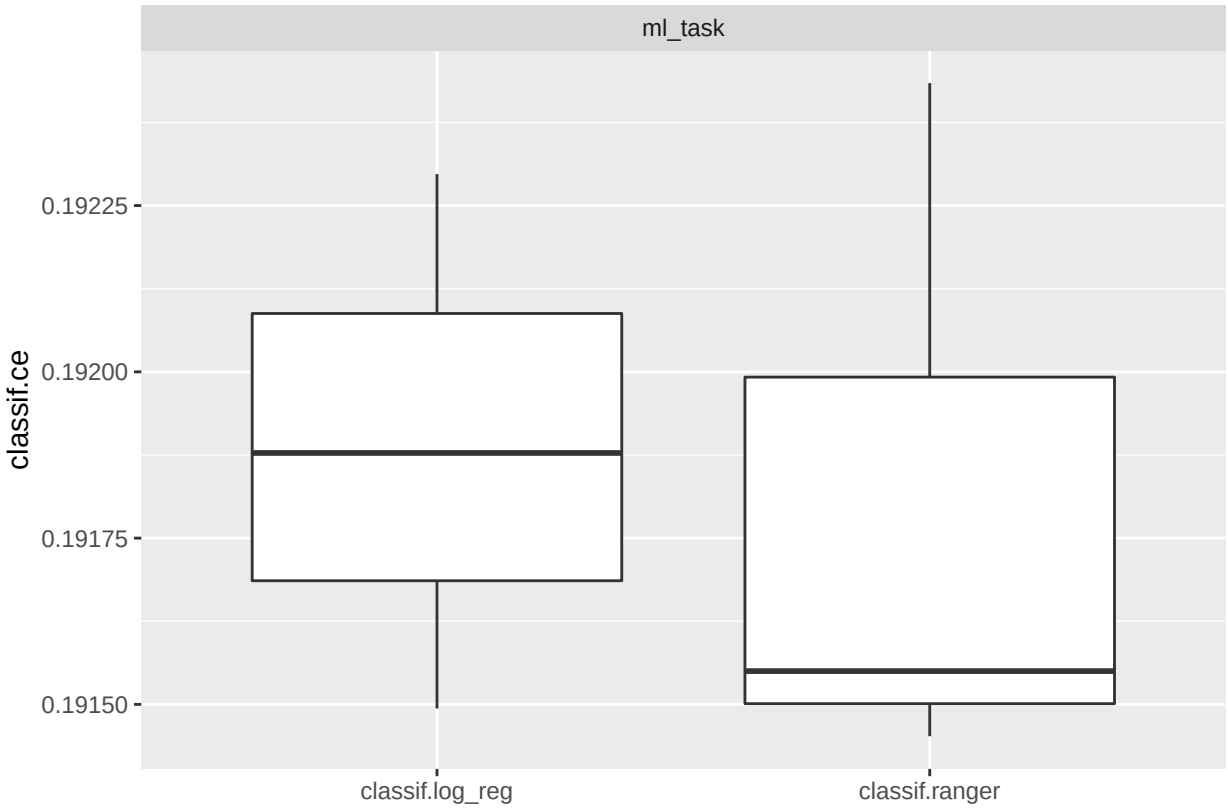
With this benchmark, we can compare different evaluation methods. Here, we focus on the `misclassfication rate` and `AUC`.

```
measures = msrs(c("classif.ce", "classif.auc"))
performances = bmr$aggregate(measures)
performances[, c("learner_id", "classif.ce", "classif.auc")]
```

```
##         learner_id classif.ce classif.auc
## 1: classif.log_reg  0.1918898   0.7144418
## 2:  classif.ranger  0.1918121   0.7190069
```

We can also utilize the `autoplot()` function implemented in the `mlr3viz` package to compare the learners' performances.

```
autoplot(bmr)
```

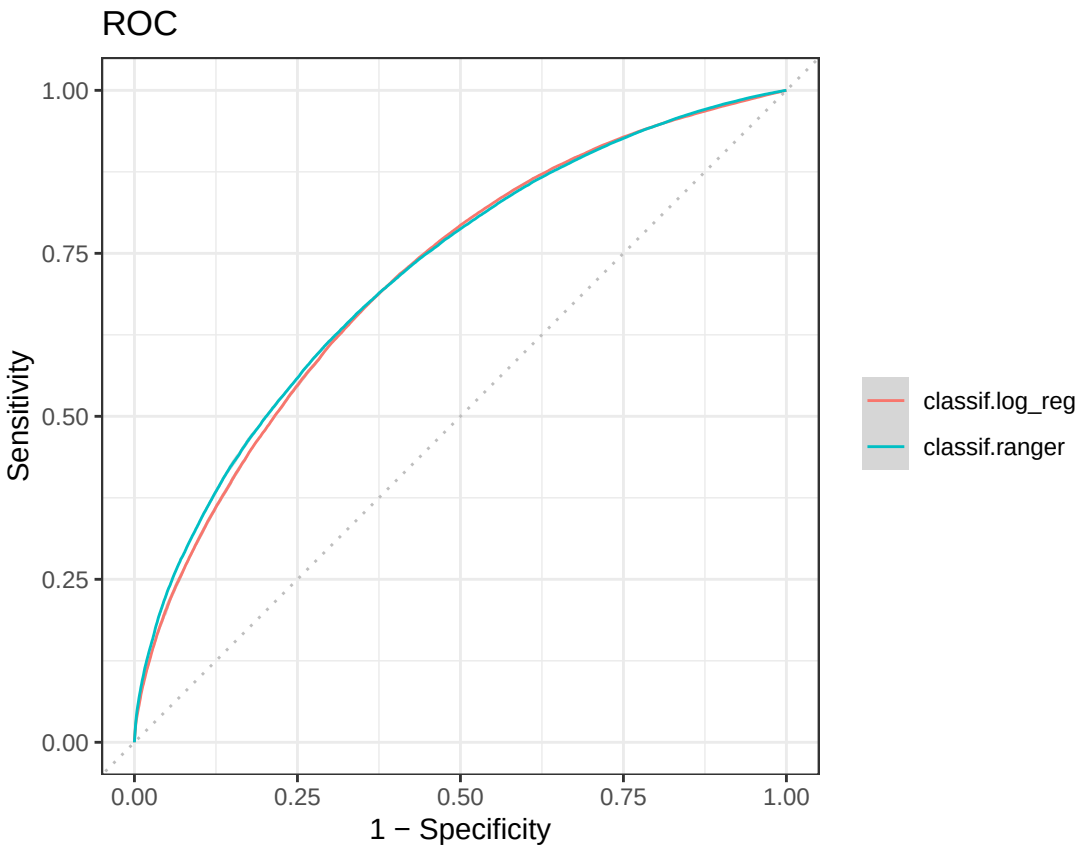


Also, we can get the ROC-curve with the following command:

```
library('precrec')
```

```
## Warning: package 'precrec' was built under R version 4.0.5
```

```
autoplot(bmr$clone(deep = TRUE)$filter(task_ids = "ml_task"), type = "roc")
```




```
DBI::dbDisconnect(conn)
```