mlr3calibration

Adrian Glos

2024-12-06

General

This package is a flexible calibration extension for the mlr3 framework in R. It includes the ability to calibrate objects of type learner, so that they can be trained or predicted as usual. It allows to choose between Platt scaling, Beta calibration and isotonic calibration. In addition, a calibrated learner can either be easily integrated into pipelines or entire pipelines can be calibrated. The calibrated learner can also be initialised by a base learner with tune token, allowing parameter tuning for the calibrated learner.

This package also provides the ability to plot the reliability curve for one or more trained learners and a task. Furthermore, this extension provides two calibration measures: the expected calibration error and the integrated calibration index.

Installation

You can install the mlr3calibration package from GitHub with the following code:

```
remotes::install_github("AdriGl117/mlr3calibration")
library(mlr3calibration)
library(mlr3verse)
```

Calibration

To use the mlr3calibration package, we first need a binary classification task

```
# Load a binary classification task
set.seed(1)
data("Sonar", package = "mlbench")
task = as_task_classif(Sonar, target = "Class", positive = "M")
splits = partition(task)
task_train = task$clone()$filter(splits$train)
task_test = task$clone()$filter(splits$test)
```

To calibrate a learner you need an uncalibrated learner, a resampling strategy and a calibration method (platt, beta or isotonic). To prevent overfitting, calibration can be performed using cross-validation. The dataset is divided into k folds, and each fold is used for calibration once, while the other (k-1) folds are used for training the classifier. If, for example, a holdout resampling strategy is selected, then no cross-validated calibration is performed, but the base learner is trained on the training split and the calibrator on the holdout.

```
# Initialize the uncalibrated learner
learner_uncal <- lrn("classif.xgboost", nrounds = 50, predict_type = "prob")

# Initialize the calibrated learner
rsmp <- rsmp("cv", folds = 5)
```

```
learner_cal <- as_learner(PipeOpCalibration$new(learner = learner_uncal,
                                                method = "beta",
                                                rsmp = rsmp))

# Set ID's for the learners
learner_uncal$id <- "Uncalibrated Learner"
learner_cal$id <- "Calibrated Learner"
```

The calibrated learner can be trained in the same way as the base learner.

```
# Train the learners
learner_uncal$train(task_train)
learner_cal$train(task_train)
```

Calibration Measures

To measure the calibration, this package provides the Expected Calibration Error (ECE) and the Integrated Calibration Index (ICI). The ECE is a measure of the difference between the predicted probabilities and the true outcomes. The ICI is a weighted average of the absolute differences between the calibration curve and the diagonal perfectly calibrated line.

```
# Predict the Learners
preds_uncal <- learner_uncal$predict(task_test)
preds_cal <- learner_cal$predict(task_test)

# Calculate the Expected Calibration Error (ECE)
ece_uncal <-  preds_uncal$score(ece$new())
ece_cal <-  preds_cal$score(ece$new())

# Uncalibrated ECE
ece_uncal
```

```
## classif.ece
##   0.1046099
```

```
# Calibrated ECE
ece_cal
```

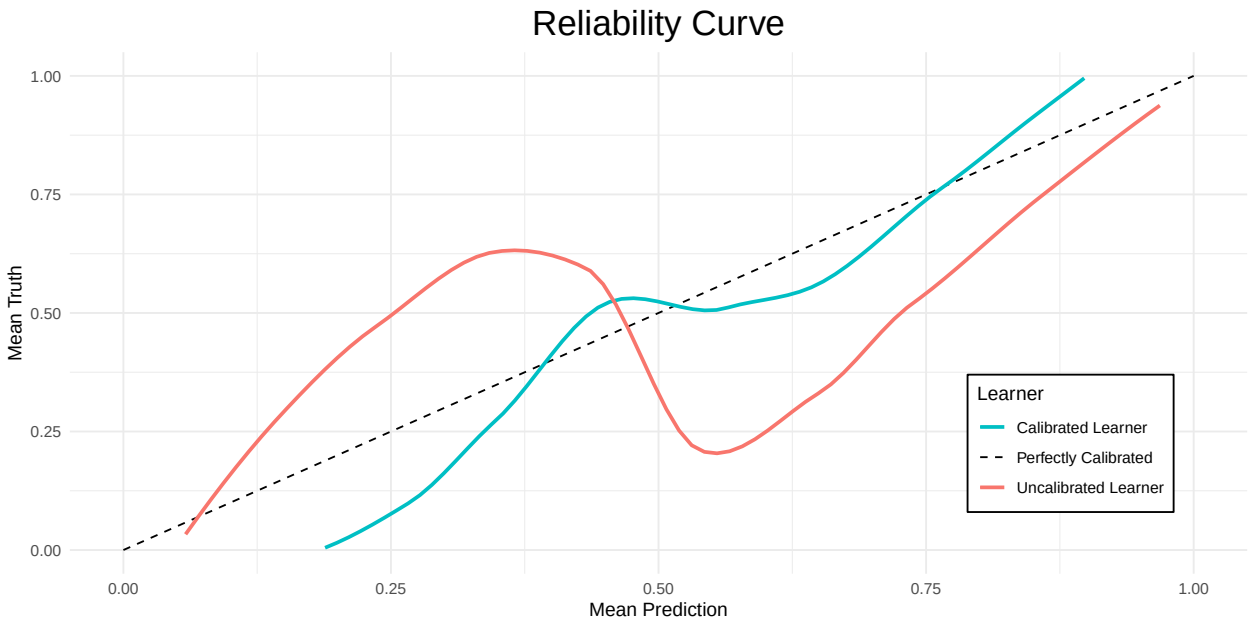
```
## classif.ece
##  0.09758731
```

Raliability Curve

To visualize how well the model is calibrated, this package provides a calibration plot. The calibration plot shows the relationship between the predicted probabilities and the true outcomes. The plot is divided into bins, and within each bin, the mean predicted probability and the mean observed outcome are calculated. The calibration plot can be smoothed using LOESS.

```
# List the Learners you want to plot
lrns = list(learner_uncal, learner_cal)

# Plot the reliability curve
calibrationplot(lrns, task_test, smooth = TRUE)
```



Calibration with resample Object

If you want to compare the three calibration approaches with each other, it can be time-consuming to train each calibration approach using cross validation. In these cases, you can train an rr object, store the models and pass it to the PipeOpCalibration. This means that the base learner has to be trained once in advance, and only the calibrators have to be trained in the PipeOpCalibration using the holdout folds.

```
# Initialize base learner
learner <- lrn("classif.xgboost", predict_type = "prob")

# Initialize resampling strategy
rsmp <- rsmp("cv", folds = 5)

# Create rr object
rr <- resample(task_train, learner, rsmp, store_models = TRUE)

# Initialze the calibrated learners
learner_calibrated_platt <- as_learner(PipeOpCalibration$new(rr = rr,
                                       method = "platt"))
learner_calibrated_platt$id = "Calibrated Platt"

learner_calibrated_beta <- as_learner(PipeOpCalibration$new(rr = rr,
                                      method = "beta"))
learner_calibrated_beta$id = "Calibrated Beta"

learner_calibrated_isotonic <- as_learner(PipeOpCalibration$new(rr = rr,
                                          method ="isotonic"))
learner_calibrated_isotonic$id = "Calibrated Isotonic"

# Train the calibrated learners
learner_calibrated_platt$train(task_train)
learner_calibrated_beta$train(task_train)
learner_calibrated_isotonic$train(task_train)
```

Pipelines

It is also possible to calibrate a mlr3 pipeline, or to include an Calibrated Learner in a pipeline.

Calibrate a Pipeline

```
# Calibrate a pipeline
pipeline <- as_learner(po("imputemean") %>>% lrn("classif.ranger",
                                                 predict_type = "prob"))

learner_cal <- as_learner(PipeOpCalibration$new(learner = pipeline,
                                    method = "platt",
                                    rsmp = rsmp("cv", folds = 5)))
learner_cal$train(task_train)
```

Include Calibrated Learner in a Pipeline

```
# Include Calibrated learner in a pipeline
learner_cal <- PipeOpCalibration$new(learner = lrn("classif.ranger",
                                                   predict_type = "prob"),
                                    rsmp = rsmp("cv", folds = 5),
                                    method = "platt")

pipeline <- as_learner(po("imputemean") %>>% learner_cal)
pipeline$train(task_train)
```

Tuning

You can either pass a tuned base learner to the calibrator or tuning your base learner within the calibrator.

```
library(mlr3tuning)
```

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

```
# Include Calibrated learner in Pipeline
learner_uncal <- lrn("classif.ranger",
                     predict_type = "prob",
                     mtry = to_tune(1, 10),
                     num.trees = to_tune(100, 500))

learner_cal <- as_learner(PipeOpCalibration$new(learner = learner_uncal,
                                                method = "platt",
                                                rsmp = rsmp("cv", folds = 5)))
at = auto_tuner(
  tuner = tnr("random_search"),
  learner = learner_cal,
  resampling = rsmp ("holdout"),
  measure = msr("classif.bbrier"),
  term_evals = 4)

at$train(task_train)
```