Evaluating Learner Performance

Table of Contents

Source: `vignettes/tutorial/performance.Rmd` (https://github.com/mlr-org/mlr/blob/master/vignettes/tutorial/performance.Rmd)

The quality of the predictions of a model in `mlr` can be assessed with respect to a number of different performance measures. In order to calculate the performance measures, call `performance() (../../reference/performance.html)` on the object returned by `predict` (`predict.WrappedModel() (../../reference/predict.WrappedModel.html)`) and specify the desired performance measures.

Available performance measures

`mlr` provides a large number of performance measures for all types of learning problems. Typical performance measures for *classification* are the mean misclassification error (mmce (measures.html)), accuracy (acc (measures.html)) or measures based on ROC analysis (roc_analysis.html). For *regression* the mean of squared errors (mse (measures.html)) or mean of absolute errors (mae (measures.html)) are usually considered. For *clustering* tasks, measures such as the Dunn index (dunn (measures.html)) are provided, while for *survival* predictions, the Concordance Index (cindex (measures.html)) is supported, and for *cost-sensitive* predictions the misclassification penalty (mcp (measures.html)) and others. It is also possible to access the time to train the learner (timetrain (measures.html)), the time to compute the prediction (timepredict (measures.html)) and their sum (timeboth (measures.html)) as performance measures.

To see which performance measures are implemented, have a look at the table of performance measures (measures.html) and the `measures()` `(../../reference/measures.html)` documentation page.

If you want to implement an additional measure or include a measure with non-standard misclassification costs, see the section on creating custom measures (create_measure.html).

Listing measures

The properties and requirements of the individual measures are shown in the table of performance measures (measures.html).

If you would like a list of available measures with certain properties or suitable for a certain learning `Task() (../../reference/Task.html)` use the function `listMeasures() (../../reference/listMeasures.html)`.

```
# Performance measures for classification with multiple classes
listMeasures (../../reference/listMeasures.html)("classif", properties = "classi
##  [1] "featperc"         "mmce"             "lsr"
##  [4] "bac"              "qsr"              "timeboth"
##  [7] "multiclass.aunp"  "timetrain"        "multiclass.aunu"
## [10] "ber"              "timepredict"      "multiclass.brier"
## [13] "ssr"              "acc"              "logloss"
## [16] "wkappa"           "multiclass.au1p"  "multiclass.au1u"
## [19] "kappa"
# Performance measure suitable for the iris classification task
listMeasures (../../reference/listMeasures.html)(iris.task)
##  [1] "featperc"         "mmce"             "lsr"
##  [4] "bac"              "qsr"              "timeboth"
##  [7] "multiclass.aunp"  "timetrain"        "multiclass.aunu"
## [10] "ber"              "timepredict"      "multiclass.brier"
## [13] "ssr"              "acc"              "logloss"
## [16] "wkappa"           "multiclass.au1p"  "multiclass.au1u"
## [19] "kappa"
```

For convenience there exists a default measure for each type of learning problem, which is calculated if nothing else is specified. As defaults we chose the most commonly used measures for the respective types, e.g., the mean squared error (mse (measures.html)) for regression and the misclassification rate (mmce (measures.html)) for classification. The help page of function `getDefaultMeasure() (../../reference/getDefaultMeasure.html)` lists all defaults for all types of learning problems. The function itself returns the default measure for a given task type, `Task() (../../reference/Task.html)` or `Learner() (../../reference/makeLearner.html)`.

```
# Get default measure for iris.task
getDefaultMeasure (../../reference/getDefaultMeasure.html)(iris.task)
## Name: Mean misclassification error
## Performance measure: mmce
## Properties: classif,classif.multi,req.pred,req.truth
## Minimize: TRUE
## Best: 0; Worst: 1
## Aggregated by: test.mean
## Arguments: list()
## Note: Defined as: mean(response != truth)

# Get the default measure for linear regression
getDefaultMeasure (../../reference/getDefaultMeasure.html)(makeLearner (../../re
## Name: Mean of squared errors
## Performance measure: mse
## Properties: regr,req.pred,req.truth
## Minimize: TRUE
## Best: 0; Worst: Inf
## Aggregated by: test.mean
## Arguments: list()
## Note: Defined as: mean((response - truth)^2)
```

Calculate performance measures

In the following example we fit a gradient boosting machine (`gbm::gbm()` (`http://www.rdocumentation.org/packages/gbm/topics/gbm`)) on a subset of the `BostonHousing` (`mlbench::BostonHousing()` (`http://www.rdocumentation.org/packages/mlbench/topics/BostonHousing`)) data set and calculate the default measure mean squared error (mse (measures.html)) on the remaining observations.

```
n = getTaskSize (../../reference/getTaskSize.html)(bh.task)
lrn = makeLearner (../../reference/makeLearner.html)("regr.gbm", n.trees = 1000)
mod = train (../../reference/train.html)(lrn, task = bh.task, subset = seq(1, n,
pred = predict(mod, task = bh.task, subset = seq(2, n, 2))

performance (../../reference/performance.html)(pred)
##      mse
## 14.38596
```

The following code computes the median of squared errors (medse (measures.html)) instead.

```
performance (../../reference/performance.html)(pred, measures = medse)
##    medse
## 4.209155
```

Of course, we can also calculate multiple performance measures at once by simply passing a list of measures which can also include your own measure (create_measure.html).

Calculate the mean squared error, median squared error and mean absolute error (mae (measures.html)).

```
performance (../../reference/performance.html)(pred, measures = list(mse, medse,
##       mse     medse       mae
## 14.385956  4.209155  2.716451
```

For the other types of learning problems and measures, calculating the performance basically works in the same way.

Requirements of performance measures

Note that in order to calculate some performance measures it is required that you pass the `Task() (../../reference/Task.html)` or the fitted model (`makeWrappedModel() (../../reference/makeWrappedModel.html)`) in addition to the `Prediction() (../../reference/Prediction.html)`.

For example in order to assess the time needed for training (timetrain (measures.html)), the fitted model has to be passed.

```
performance (../../reference/performance.html)(pred, measures = timetrain, model
## timetrain
##     0.055
```

For many performance measures in cluster analysis the `Task() (../../reference/Task.html)` is required.

```
lrn = makeLearner (../../reference/makeLearner.html)("cluster.kmeans", centers =
mod = train (../../reference/train.html)(lrn, mtcars.task)
pred = predict(mod, task = mtcars.task)

# Calculate the Dunn index
performance (../../reference/performance.html)(pred, measures = dunn, task = mtca
##      dunn
## 0.1462919
```

Moreover, some measures require a certain type of prediction. For example in binary classification in order to calculate the AUC (auc (measures.html)) – the area under the ROC (receiver operating characteristic) curve – we have to make sure that posterior probabilities are predicted. For more information on ROC analysis, see the section on ROC analysis (roc_analysis.html).

```
lrn = makeLearner (../../reference/makeLearner.html)("classif.rpart", predict.typ
mod = train (../../reference/train.html)(lrn, task = sonar.task)
pred = predict(mod, task = sonar.task)

performance (../../reference/performance.html)(pred, measures = auc)
##       auc
## 0.9224018
```

Also bear in mind that many of the performance measures that are available for classification, e.g., the false positive rate (fpr (measures.html)), are only suitable for binary problems.

Access a performance measure

Performance measures in `mlr` are objects of class `Measure` (`makeMeasure()` `(../../reference/makeMeasure.html)`). If you are interested in the properties or requirements of a single measure you can access it directly. See the help page of `Measure` (`makeMeasure()` `(../../reference/makeMeasure.html)`) for information on the individual slots.

```
# Mean misclassification error
str(mmce)
## List of 10
##  $ id        : chr "mmce"
##  $ minimize  : logi TRUE
##  $ properties: chr [1:4] "classif" "classif.multi" "req.pred" "req.truth"
##  $ fun       :function (task, model, pred, feats, extra.args)
##  $ extra.args: list()
##  $ best      : num 0
##  $ worst     : num 1
##  $ name      : chr "Mean misclassification error"
##  $ note      : chr "Defined as: mean(response ≠ truth)"
##  $ aggr      :List of 4
##   ..$ id        : chr "test.mean"
##   ..$ name      : chr "Test mean"
##   ..$ fun       :function (task, perf.test, perf.train, measure, group, pred)
##   ..$ properties: chr "req.test"
##   ..- attr(*, "class")= chr "Aggregation"
##  - attr(*, "class")= chr "Measure"
```

Binary classification

For binary classification specialized techniques exist to analyze the performance.

Plot performance versus threshold

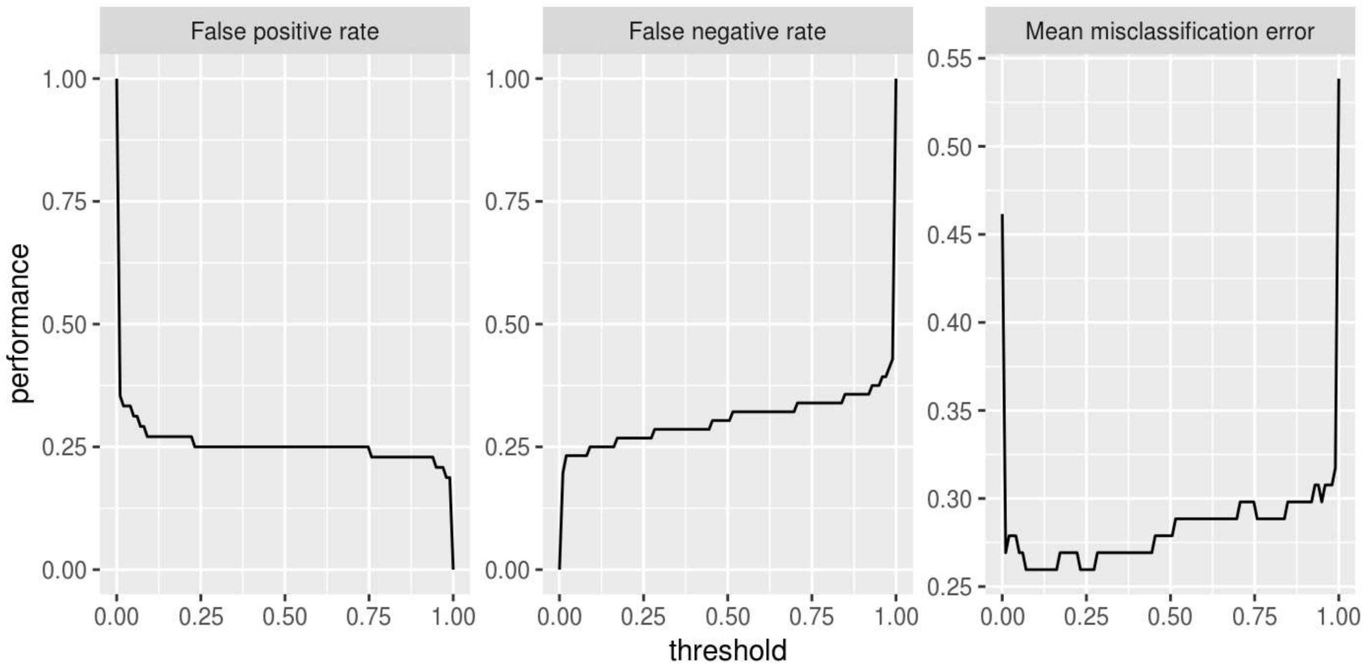
As you may recall (see the previous section on making predictions (predict.html)) in binary classification we can adjust the threshold used to map probabilities to class labels. Helpful in this regard is are the functions `generateThreshVsPerfData()` `(../../reference/generateThreshVsPerfData.html)` and `plotThreshVsPerf()` `(../../reference/plotThreshVsPerf.html)`, which generate and plot, respectively, the learner performance versus the threshold.

For more performance plots and automatic threshold tuning see the section on ROC analysis (roc_analysis.html).

In the following example we consider the `mlbench::Sonar()` `(http://www.rdocumentation.org/packages/mlbench/topics/Sonar)` data set and plot the false positive rate (fpr (measures.html)), the false negative rate (fnr (measures.html)) as well as the misclassification rate (mmce (measures.html)) for all possible threshold values.

```
lrn = makeLearner (../../reference/makeLearner.html)("classif.lda", predict.type
n = getTaskSize (../../reference/getTaskSize.html)(sonar.task)
mod = train (../../reference/train.html)(lrn, task = sonar.task, subset = seq(1,
pred = predict(mod, task = sonar.task, subset = seq(2, n, by = 2))

# Performance for the default threshold 0.5
performance (../../reference/performance.html)(pred, measures = list(fpr, fnr, mm
##       fpr       fnr      mmce
## 0.2500000 0.3035714 0.2788462
# Plot false negative and positive rates as well as the error rate versus the thr
d = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(pred
plotThreshVsPerf (../../reference/plotThreshVsPerf.html)(d)
```



There is an experimental `ggvis` plotting function `plotThreshVsPerfGGVIS()` which performs similarly to `plotThreshVsPerf()` `(../../reference/plotThreshVsPerf.html)` but instead of creating facetted subplots to visualize multiple learners and/or multiple measures, one of them is mapped to an interactive sidebar which selects what to display.

```
plotThreshVsPerfGGVIS(d)
```

ROC measures

For binary classification a large number of specialized measures exist, which can be nicely formatted into one matrix, see for example the receiver operating characteristic page on wikipedia (https://en.wikipedia.org/wiki/Receiver_operating_characteristic).

We can generate a similiar table with the `calculateROCMeasures()` `(../../reference/calculateROCMeasures.html)` function.

```
r = calculateROCMeasures (../../reference/calculateROCMeasures.html)(pred)
r
##     predicted
## true M         R
##    M 39        17        tpr: 0.7  fnr: 0.3
##    R 12        36        fpr: 0.25 tnr: 0.75
##      ppv: 0.76 for: 0.32 lrp: 2.79 acc: 0.72
##      fdr: 0.24 npv: 0.68 lrm: 0.4  dor: 6.88
##
##
## Abbreviations:
## tpr - True positive rate (Sensitivity, Recall)
## fpr - False positive rate (Fall-out)
## fnr - False negative rate (Miss rate)
## tnr - True negative rate (Specificity)
## ppv - Positive predictive value (Precision)
## for - False omission rate
## lrp - Positive likelihood ratio (LR+)
## fdr - False discovery rate
## npv - Negative predictive value
## acc - Accuracy
## lrm - Negative likelihood ratio (LR-)
## dor - Diagnostic odds ratio
```

The top left 2×2 matrix is the confusion matrix (predict.html), which shows the relative frequency of correctly and incorrectly classified observations. Below and to the right a large number of performance measures that can be inferred from the confusion matrix are added. By default some additional info about the measures is printed. You can turn this off using the `abbreviations` argument of the `print` (`calculateROCMeasures()` `(../../reference/calculateROCMeasures.html)`) method: `print(r, abbreviations = FALSE)`.