ROC Analysis and Performance Curves

Table of Contents

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

For binary scoring classifiers a *threshold* (or *cutoff*) value controls how predicted posterior probabilities are converted into class labels. ROC curves and other performance plots serve to visualize and analyse the relationship between one or two performance measures and the threshold.

This page is mainly devoted to *receiver operating characteristic* (ROC) curves that plot the *true positive rate* (sensitivity) on the vertical axis against the *false positive rate* (1 - specificity, fall-out) on the horizontal axis for all possible threshold values. Creating other performance plots like *lift charts* or *precision/recall graphs* works analogously and is shown briefly.

In addition to performance visualization ROC curves are helpful in

- determining an optimal decision threshold for given class prior probabilities and misclassification costs (for alternatives see also the pages about cost-sensitive classification (cost_sensitive_classif.html) and imbalanced classification problems (over_and_undersampling.html) in this tutorial),
- identifying regions where one classifier outperforms another and building suitable multi-classifier systems,
- obtaining calibrated estimates of the posterior probabilities.

For more information see the tutorials and introductory papers by Fawcett (2004) (http://binf.gmu.edu/mmasso/ROC101.pdf), Fawcett (2006) (https://ccrma.stanford.edu/workshops/mir2009/references/ROCintro.pdf) as well as Flach (ICML 2004) (http://www.cs.bris.ac.uk/~flach/ICML04tutorial/index.html).

In many applications as, e.g., diagnostic tests or spam detection, there is uncertainty about the class priors or the misclassification costs at the time of prediction, for example because it's hard to quantify the costs or because costs and class priors vary over time. Under these circumstances the classifier is expected to work well for a whole range of decision thresholds and the area under the ROC curve (AUC) provides a scalar performance measure for comparing and selecting classifiers. `mlr` provides the AUC for binary classification (auc (measures.html)) and also several generalizations of the AUC to the multi-class case (e.g., multiclass.au1p (measures.html), multiclass.au1u (measures.html) based on Ferri et al. (2009) (https://www.math.ucdavis.edu/~saito/data/roc/ferri-class-perf-metrics.pdf)).

`mlr` offers three ways to plot ROC and other performance curves.

1. Function `plotROCCurves() (../../reference/plotROCCurves.html)` can, based on the output of `generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)`, plot performance curves for any pair of performance measures (measures.html) available in `mlr`.
2. `mlr` offers an interface to package `ROCR` through function `asROCRPrediction() (../../reference/asROCRPrediction.html)`.
3. `mlr`'s function `plotViperCharts()` provides an interface to ViperCharts (http://viper.ijs.si).

With `mlr` version 2.8 functions `generateROCRCurvesData`, `plotROCRCurves`, and `plotROCRCurvesGGVIS` were deprecated.

Below are some examples that demonstrate the three possible ways. Note that you can only use learners (learner.html) that are capable of predicting probabilities. Have a look at the learner table in the Appendix (integrated_learners.html) or run `listLearners("classif", properties = c("twoclass", "prob")) (../../reference/listLearners.html)` to get a list of all learners that support this.

Performance plots with plotROCCurves

As you might recall `generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)` calculates one or several performance measures for a sequence of decision thresholds from 0 to 1. It provides S3 methods for objects of class `Prediction() (../../reference/Prediction.html)`, `ResampleResult() (../../reference/ResampleResult.html)` and `BenchmarkResult() (../../reference/BenchmarkResult.html)` (resulting from `predict` (`predict.WrappedModel() (../../reference/predict.WrappedModel.html)`), `resample() (../../reference/resample.html)` or `benchmark() (../../reference/benchmark.html)`). `plotROCCurves() (../../reference/plotROCCurves.html)` plots the result of `generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)` using `ggplot2`.

Example 1: Single predictions

We consider the `Sonar` (`mlbench::Sonar() (http://www.rdocumentation.org/packages/mlbench/topics/Sonar)`) data set from package `mlbench`, which poses a binary classification problem (`sonar.task() (../../reference/sonar.task.html)`) and apply linear discriminant analysis (`MASS::lda() (http://www.rdocumentation.org/packages/MASS/topics/lda)`).

```
n = getTaskSize (../../reference/getTaskSize.html)(sonar.task)
train.set = sample(n, size = round(2/3 * n))
test.set = setdiff(seq_len(n), train.set)

lrn1 = makeLearner (../../reference/makeLearner.html)("classif.lda", predict.type
mod1 = train (../../reference/train.html)(lrn1, sonar.task, subset = train.set)
pred1 = predict(mod1, task = sonar.task, subset = test.set)
```

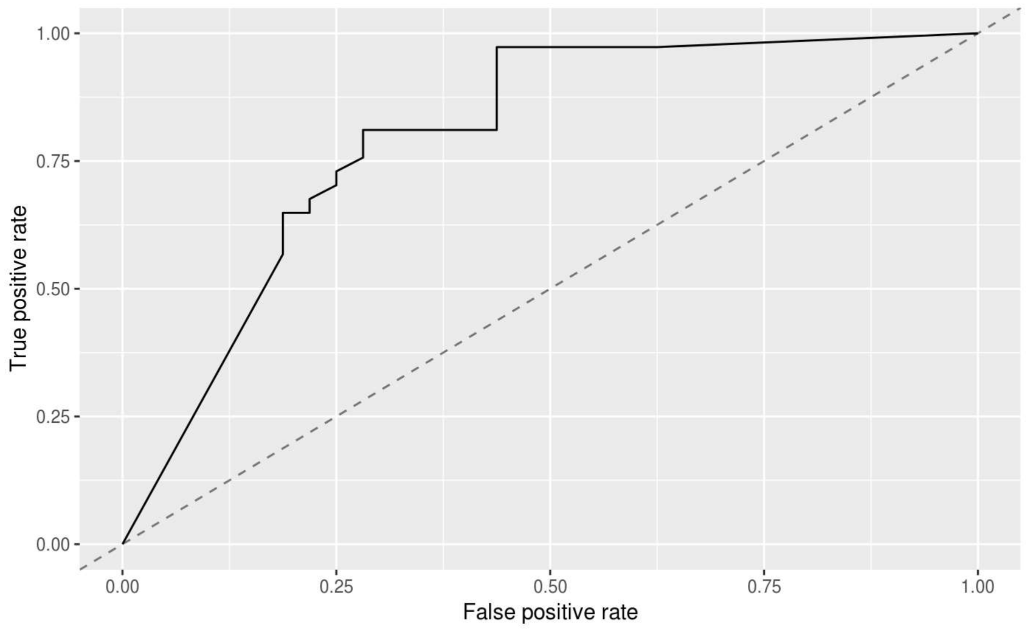
Since we want to plot ROC curves we calculate the false and true positive rates (fpr (measures.html) and tpr (measures.html)). Additionally, we also compute error rates (mmce (measures.html)).

```
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(pre
```

`generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)` returns an object of class `ThreshVsPerfData` (`generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)`) which contains the performance values in the `$data` element.

Per default, `plotROCCurves() (../../reference/plotROCCurves.html)` plots the performance values of the first two measures passed to `generateThreshVsPerfData() (../../reference/generateThreshVsPerfData.html)`. The first is shown on the x-axis, the second on the y-axis. Moreover, a diagonal line that represents the performance of a random classifier is added. You can remove the diagonal by setting `diagonal = FALSE`.

```
plotROCCurves (../../reference/plotROCCurves.html)(df)
```

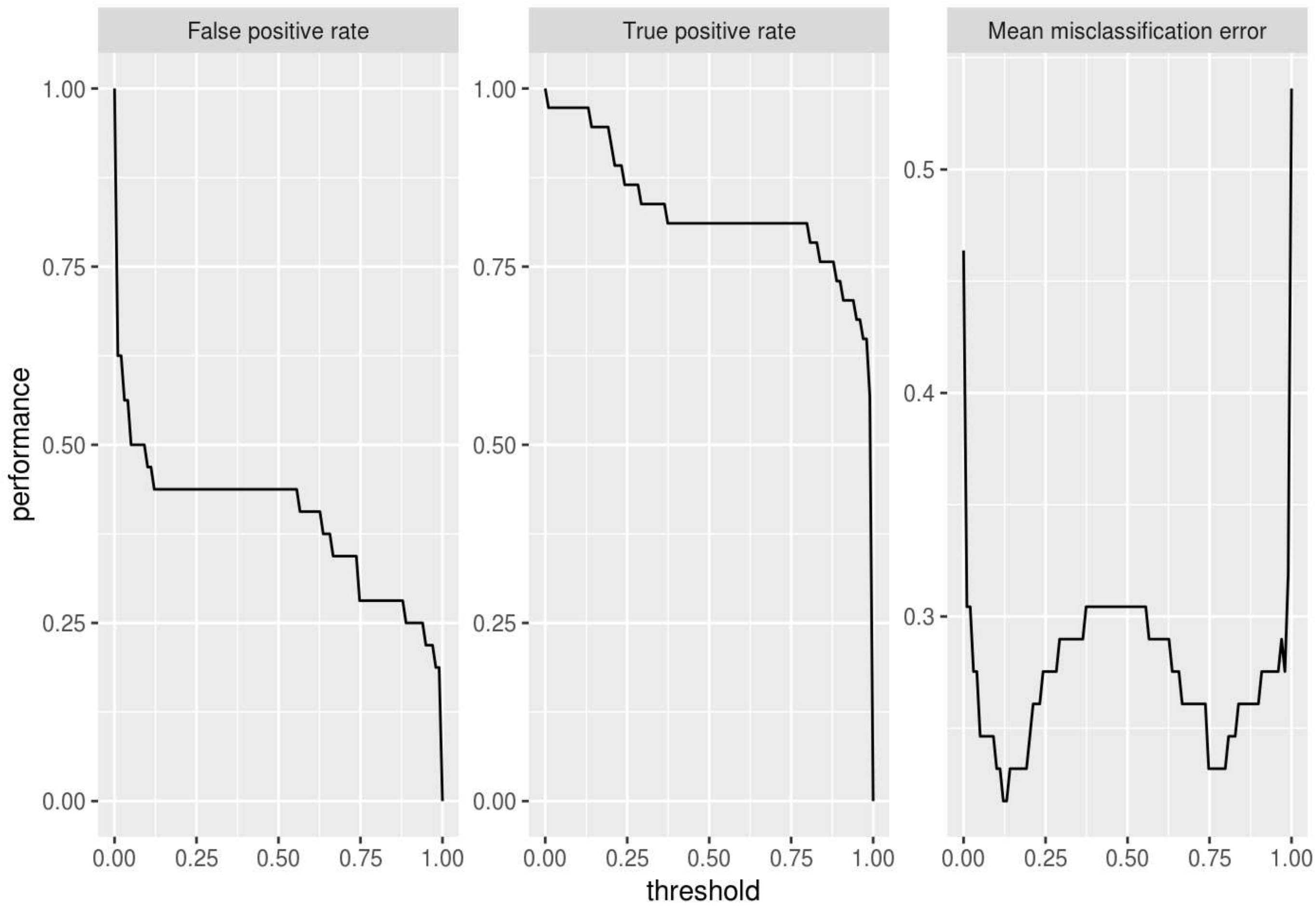


The corresponding area under curve (auc (measures.html)) can be calculated as usual by calling `performance() (../../reference/performance.html)`.

```
mlr::performance (http://www.rdocumentation.org/packages/mlr/topics/performance)(
##       auc
## 0.7930743
```

`plotROCCurves() (../../reference/plotROCCurves.html)` always requires a pair of performance measures that are plotted against each other. If you want to plot individual measures versus the decision threshold you can use function `plotThreshVsPerf() (../../reference/plotThreshVsPerf.html)`.

```
plotThreshVsPerf (../../reference/plotThreshVsPerf.html)(df)
```



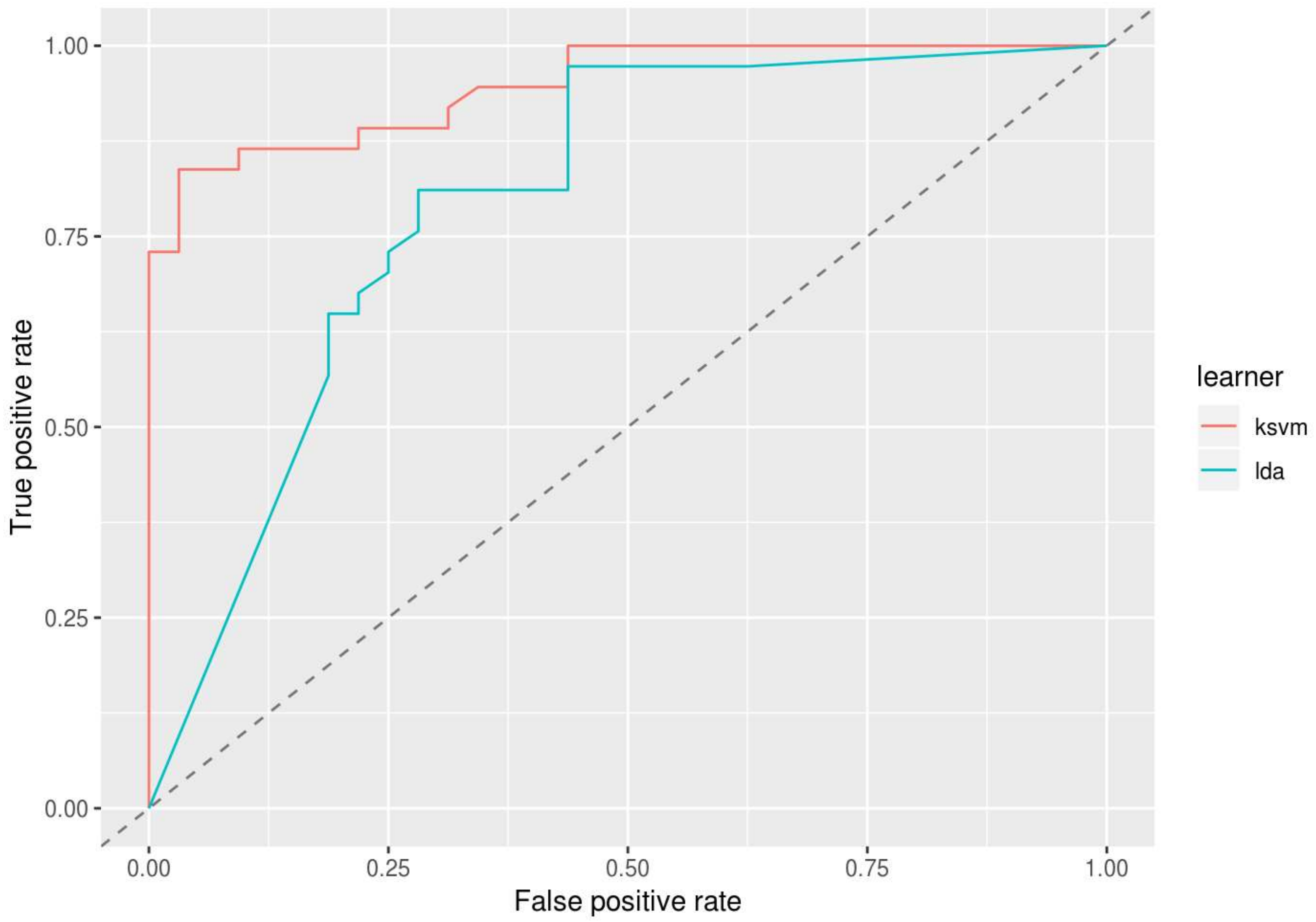
Additional to linear discriminant analysis (`MASS::lda() (http://www.rdocumentation.org/packages/MASS/topics/lda)`) we try a support vector machine with RBF kernel (`kernlab::ksvm() (http://www.rdocumentation.org/packages/kernlab/topics/ksvm)`).

```
lrn2 = makeLearner (../../reference/makeLearner.html)("classif.ksvm", predict.typ
mod2 = train (../../reference/train.html)(lrn2, sonar.task, subset = train.set)
pred2 = predict(mod2, task = sonar.task, subset = test.set)
```

In order to compare the performance of the two learners you might want to display the two corresponding ROC curves in one plot. For this purpose just pass a named `list` of `Prediction() (../../reference/Prediction.html)` s to `generateThreshVsPerfData()`

(../../reference/generateThreshVsPerfData.html).

```
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(lis
plotROCCurves (../../reference/plotROCCurves.html)(df)
```

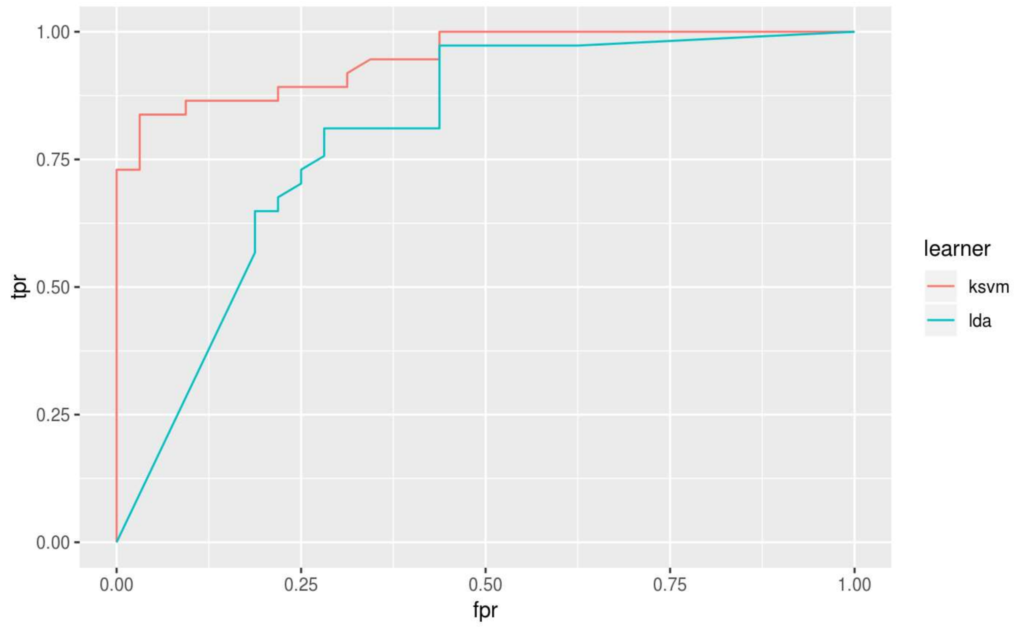


It's clear from the plot above that `kernlab::ksvm()` (http://www.rdocumentation.org/packages/kernlab/topics/ksvm) has a slightly higher AUC than lda (`MASS::lda()` (http://www.rdocumentation.org/packages/MASS/topics/lda)).

```
mlr::performance (http://www.rdocumentation.org/packages/mlr/topics/performance)(
##       auc
## 0.9467905
```

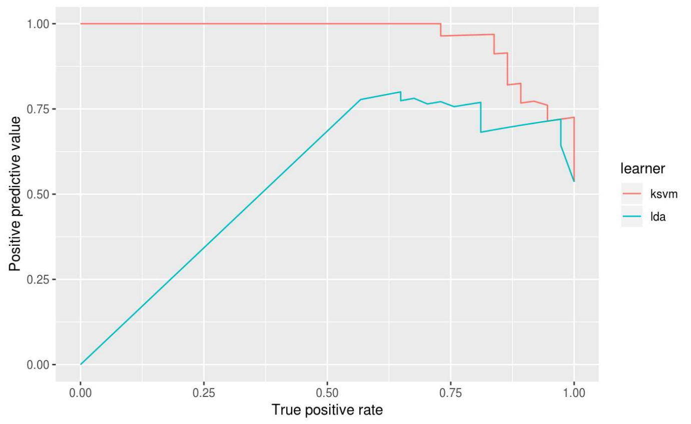
Based on the `$data` member of `df` you can easily generate custom plots. Below the curves for the two learners are superposed.

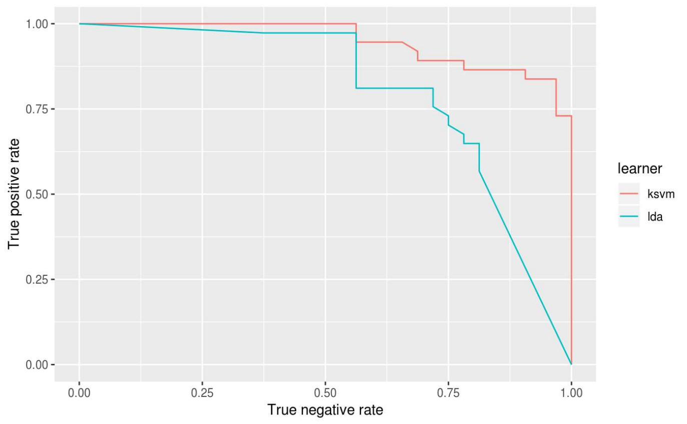
```
qplot(x = fpr, y = tpr, color = learner, data = df$data, geom = "path")
```



It is easily possible to generate other performance plots by passing the appropriate performance measures to `generateThreshVsPerfData()` (`../../reference/generateThreshVsPerfData.html`) and `plotROCCurves()` (`../../reference/plotROCCurves.html`). Below, we generate a *precision/recall graph* (precision = positive predictive value = ppv, recall = tpr) and a *sensitivity/specificity plot* (sensitivity = tpr, specificity = tnr).

```
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(lis

# Precision/recall graph
plotROCCurves (../../reference/plotROCCurves.html)(df, measures = list(tpr, ppv)

# Sensitivity/specificity plot
plotROCCurves (../../reference/plotROCCurves.html)(df, measures = list(tnr, tpr)
```





Example 2: Benchmark experiment

The analysis in the example above can be improved a little. Instead of writing individual code for training/prediction of each learner, which can become tedious very quickly, we can use function `benchmark()` (`../../reference/benchmark.html`) (see also Benchmark

Experiments (benchmark_experiments.html)) and, ideally, the support vector machine should have been tuned (tune.html).

We again consider the `Sonar` (`mlbench::Sonar()` `(http://www.rdocumentation.org/packages/mlbench/topics/Sonar)`) data set and apply `MASS::lda() (http://www.rdocumentation.org/packages/MASS/topics/lda)` as well as `kernlab::ksvm() (http://www.rdocumentation.org/packages/kernlab/topics/ksvm)`. We first generate a tuning wrapper (`makeTuneWrapper()` `(../../reference/makeTuneWrapper.html)`) for `kernlab::ksvm()` `(http://www.rdocumentation.org/packages/kernlab/topics/ksvm)`. The cost parameter is tuned on a (for demonstration purposes small) parameter grid. We assume that we are interested in a good performance over the complete threshold range and therefore tune with regard to the auc (measures.html). The error rate (mmce (measures.html)) for a threshold value of 0.5 is reported as well.

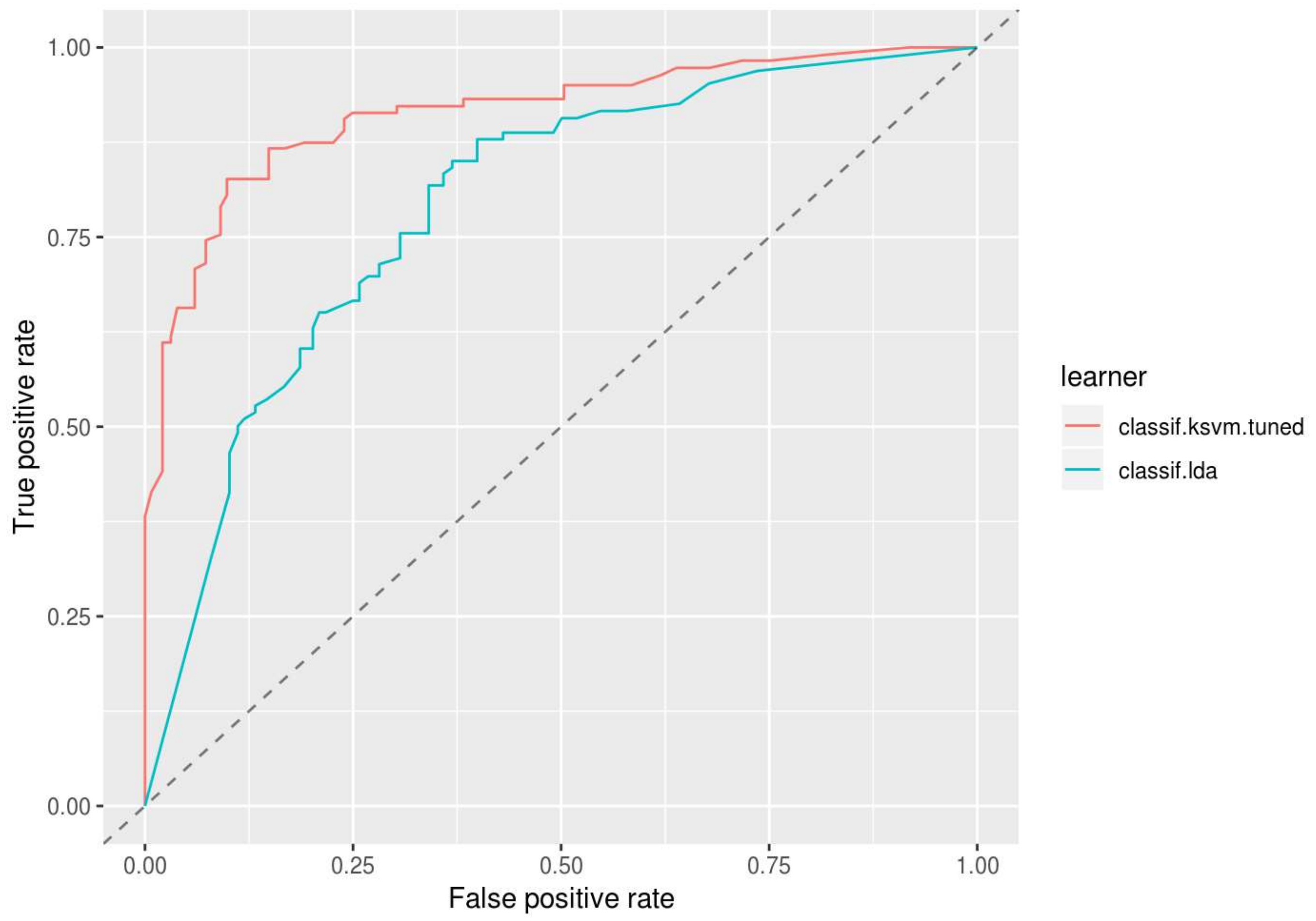
```
# Tune wrapper for ksvm
rdesc.inner = makeResampleDesc (../../reference/makeResampleDesc.html)("Holdout"
ms = list(mlr::auc, mmce)
ps = makeParamSet(
  makeDiscreteParam("C", 2^(-1:1))
)
ctrl = makeTuneControlGrid (../../reference/makeTuneControlGrid.html)()
lrn2 = makeTuneWrapper (../../reference/makeTuneWrapper.html)(lrn2, rdesc.inner,
```

Below the actual benchmark experiment is conducted. As resampling strategy we use 5-fold cross-validation and again calculate the auc (measures.html) as well as the error rate (for a threshold/cutoff value of 0.5).

```
# Benchmark experiment
lrns = list(lrn1, lrn2)
rdesc.outer = makeResampleDesc (../../reference/makeResampleDesc.html)("CV", ite

bmr = benchmark (../../reference/benchmark.html)(lrns, tasks = sonar.task, resam
bmr
##         task.id         learner.id auc.test.mean mmce.test.mean
## 1 Sonar-example        classif.lda     0.7999941      0.2792102
## 2 Sonar-example classif.ksvm.tuned     0.9130424      0.1635308
```

Calling `generateThreshVsPerfData()` `(../../reference/generateThreshVsPerfData.html)` and `plotROCCurves()` `(../../reference/plotROCCurves.html)` on the benchmark result (`BenchmarkResult()` `(../../reference/BenchmarkResult.html)`) produces a plot with ROC curves for all learners in the experiment.

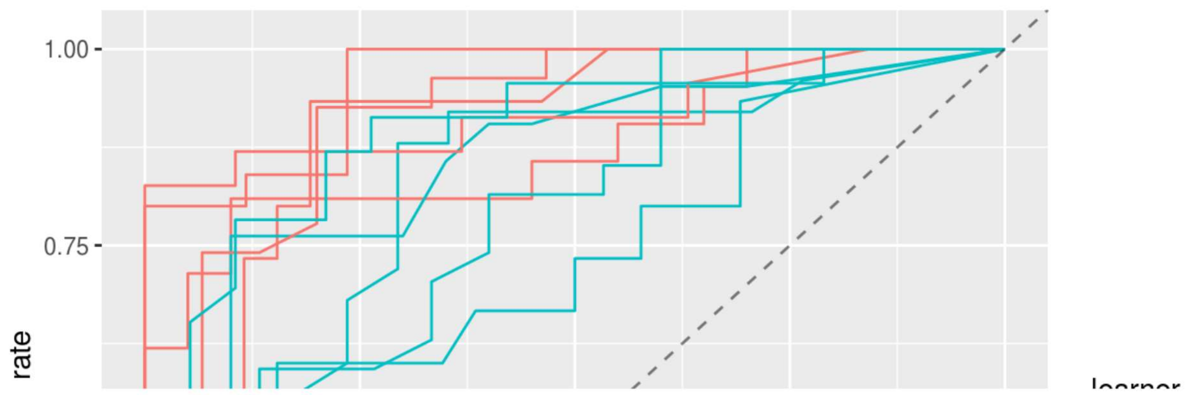
```
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(bm
plotROCCurves (../../reference/plotROCCurves.html)(df)
```



Per default, `generateThreshVsPerfData()` (`../../reference/generateThreshVsPerfData.html`) calculates aggregated performances according to the chosen resampling strategy (5-fold cross-validation) and aggregation scheme (`test.mean` (`aggregations()` (`../../reference/aggregations.html`))) for each threshold in the sequence. This way we get *threshold-averaged* ROC curves.

If you want to plot the individual ROC curves for each resample iteration set `aggregate = FALSE`.

```
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(bmr
plotROCCurves (../../reference/plotROCCurves.html)(df)
```



The same applies for `plotThreshVsPerf()` `(../../reference/plotThreshVsPerf.html)`.

```
plotThreshVsPerf (../../reference/plotThreshVsPerf.html)(df) +
  theme(strip.text.x = element_text(size = 7))
```

An alternative to averaging is to just merge the 5 test folds and draw a single ROC curve. Merging can be achieved by manually changing the class attribute of the prediction objects from `ResamplePrediction()` `(../../reference/ResamplePrediction.html)` to `Prediction()` `(../../reference/Prediction.html)`.

Below, the predictions are extracted from the `BenchmarkResult()` `(../../reference/BenchmarkResult.html)` via function `getBMRPredictions()` `(../../reference/getBMRPredictions.html)`, the class is changed and the ROC curves are created.

Averaging methods are normally preferred (cp. Fawcett, 2006 (https://ccrma.stanford.edu/workshops/mir2009/references/ROCintro.pdf)), as they permit to assess the variability, which is needed to properly compare classifier performance.

```
# Extract predictions
preds = getBMRPredictions (../../reference/getBMRPredictions.html)(bmr, drop = TR

# Change the class attribute
preds2 = lapply(preds, function(x) {class(x) = "Prediction"; return(x)})

# Draw ROC curves
df = generateThreshVsPerfData (../../reference/generateThreshVsPerfData.html)(pre
plotROCCurves (../../reference/plotROCCurves.html)(df)
```

Again, you can easily create other standard evaluation plots by passing the appropriate performance measures to `generateThreshVsPerfData()` `(../../reference/generateThreshVsPerfData.html)` and `plotROCCurves()` `(../../reference/plotROCCurves.html)`.

Performance plots with asROCRPrediction

Drawing performance plots with package `ROCR` works through three basic commands:

1. `ROCR::prediction()` `(http://www.rdocumentation.org/packages/ROCR/topics/prediction)`: Create a `ROCR` prediction object.
2. `ROCR::performance()` `(http://www.rdocumentation.org/packages/ROCR/topics/performance)`: Calculate one or more performance measures for the given prediction object.
3. `ROCR::plot()`: Generate the performance plot.

`mlr`'s function `asROCRPrediction()` `(../../reference/asROCRPrediction.html)` converts an `mlr` `Prediction()` `(../../reference/Prediction.html)` object to a `ROCR` prediction (`ROCR::prediction-class()`) object, so you can easily generate performance plots by doing steps 2. and 3. yourself. `ROCR`'s plot (`ROCR::plot-methods()`) method has some nice features which are not (yet) available in `plotROCCurves()` `(../../reference/plotROCCurves.html)`, for example plotting the convex hull of the ROC curves. Some examples are shown below.

Example 1: Single predictions (continued)

We go back to out first example where we trained and predicted `MASS::lda()` `(http://www.rdocumentation.org/packages/MASS/topics/lda)` on the sonar classification task (`sonar.task()` `(../../reference/sonar.task.html)`).

```
n = getTaskSize (../../reference/getTaskSize.html)(sonar.task)
train.set = sample(n, size = round(2/3 * n))
test.set = setdiff(seq_len(n), train.set)

# Train and predict linear discriminant analysis
lrn1 = makeLearner (../../reference/makeLearner.html)("classif.lda", predict.type
mod1 = train (../../reference/train.html)(lrn1, sonar.task, subset = train.set)
pred1 = predict(mod1, task = sonar.task, subset = test.set)
```

Below we use `asROCRPrediction()` `(../../reference/asROCRPrediction.html)` to convert the lda prediction, let `ROCR` calculate the true and false positive rate and plot the ROC curve.

```
# Convert prediction
ROCRpred1 = asROCRPrediction (../../reference/asROCRPrediction.html)(pred1)

# Calculate true and false positive rate
ROCRperf1 = ROCR::performance (http://www.rdocumentation.org/packages/ROCR/topics

# Draw ROC curve
ROCR::plot(ROCRperf1)
```

Below is the same ROC curve, but we make use of some more graphical parameters: The ROC curve is color-coded by the threshold and selected threshold values are printed on the curve. Additionally, the convex hull (black broken line) of the ROC curve is drawn.

```
# Draw ROC curve
ROCR::plot(ROCRperf1, colorize = TRUE, print.cutoffs.at = seq(0.1, 0.9, 0.1), lwd

# Draw convex hull of ROC curve
ch = ROCR::performance (http://www.rdocumentation.org/packages/ROCR/topics/perfor
ROCR::plot(ch, add = TRUE, lty = 2)
```

Example 2: Benchmark experiments (continued)

We again consider the benchmark experiment conducted earlier. We first extract the predictions by `getBMRPredictions()` `(../../reference/getBMRPredictions.html)` and then convert them via function `asROCRPrediction()` `(../../reference/asROCRPrediction.html)`.

```
# Extract predictions
preds = getBMRPredictions (../../reference/getBMRPredictions.html)(bmr, drop = TR

# Convert predictions
ROCRpreds = lapply(preds, asROCRPrediction)

# Calculate true and false positive rate
ROCRperfs = lapply(ROCRpreds, function(x) ROCR::performance (http://www.rdocument
```

We draw the vertically averaged ROC curves (solid lines) as well as the ROC curves for the individual resampling iterations (broken lines). Moreover, standard error bars are plotted for selected true positive rates (0.1, 0.2, ..., 0.9). See `ROCR`'s plot (`ROCR::plot-methods()`) function for details.

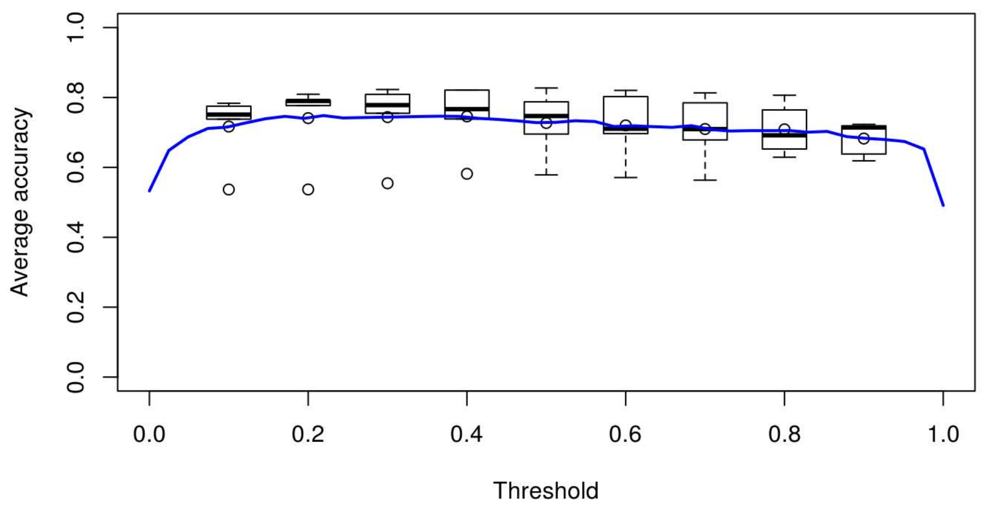
```
# lda average ROC curve
plot(ROCRperfs[[1]], col = "blue", avg = "vertical", spread.estimate = "stderror"
  show.spread.at = seq(0.1, 0.8, 0.1), plotCI.col = "blue", plotCI.lwd = 2, lwd =
# lda individual ROC curves
plot(ROCRperfs[[1]], col = "blue", lty = 2, lwd = 0.25, add = TRUE)

# ksvm average ROC curve
plot(ROCRperfs[[2]], col = "red", avg = "vertical", spread.estimate = "stderror",
  show.spread.at = seq(0.1, 0.6, 0.1), plotCI.col = "red", plotCI.lwd = 2, lwd =
# ksvm individual ROC curves
plot(ROCRperfs[[2]], col = "red", lty = 2, lwd = 0.25, add = TRUE)

legend("bottomright", legend = getBMRLearnerIds (../../reference/getBMRLearnerIds
```

In order to create other evaluation plots like *precision/recall graphs* you just have to change the performance measures when calling `ROCR::performance()` `(http://www.rdocumentation.org/packages/ROCR/topics/performance)`. (Note that you have to use the measures provided by `ROCR` listed in `ROCR::performance()` `(http://www.rdocumentation.org/packages/ROCR/topics/performance)` and not `mlr`'s performance measures.)

```
# Extract and convert predictions
preds = getBMRPredictions (../../reference/getBMRPredictions.html)(bmr, drop = TF
ROCRpreds = lapply(preds, asROCRPrediction)

# Calculate precision and recall
ROCRperfs = lapply(ROCRpreds, function(x) ROCR::performance (http://www.rdocument

# Draw performance plot
plot(ROCRperfs[[1]], col = "blue", avg = "threshold")
plot(ROCRperfs[[2]], col = "red", avg = "threshold", add = TRUE)
legend("bottomleft", legend = getBMRLearnerIds (../../reference/getBMRLearnerIds.
```

If you want to plot a performance measure versus the threshold, specify only one measure when calling `ROCR::performance()` `(http://www.rdocumentation.org/packages/ROCR/topics/performance)`. Below the average accuracy over the 5 cross-validation iterations is plotted against the threshold. Moreover, boxplots for certain threshold values (0.1, 0.2, ..., 0.9) are drawn.

```
# Extract and convert predictions
preds = getBMRPredictions (../../reference/getBMRPredictions.html)(bmr, drop = TF
ROCRpreds = lapply(preds, asROCRPrediction)

# Calculate accuracy
ROCRperfs = lapply(ROCRpreds, function(x) ROCR::performance (http://www.rdocument

# Plot accuracy versus threshold
plot(ROCRperfs[[1]], avg = "vertical", spread.estimate = "boxplot", lwd = 2, col
  show.spread.at = seq(0.1, 0.9, 0.1), ylim = c(0,1), xlab = "Threshold")
```



Viper charts

`mlr` also supports ViperCharts (http://viper.ijs.si/) for plotting ROC and other performance curves. Like `generateThreshVsPerfData()` `(../../reference/generateThreshVsPerfData.html)` it has S3 methods for objects of class `Prediction()` `(../../reference/Prediction.html)`, `ResampleResult()` `(../../reference/ResampleResult.html)` and `BenchmarkResult()` `(../../reference/BenchmarkResult.html)`. Below plots for the benchmark experiment (Example 2) are generated.

```
z = plotViperCharts(bmr, chart = "rocc", browse = FALSE)
## Error in plotViperCharts(bmr, chart = "rocc", browse = FALSE): could not find
```

Note that besides ROC curves you get several other plots like lift charts or cost curves. For details, see `plotViperCharts()`.