Example workflow using OpenML and mlr

The OpenML R Team

2016-03-18

In this vignette we illustrate the advantages of OpenML by performing a small comparison study between a random forest and bagged trees. We first create the respective binary classification learners using mlr, then query OpenML for suitable data sets, apply the learners on the data, and finally evaluate the results.

Create Learners

Because there is a variety of tree implementations in R, we select three implementations of different algorithms. These are **rpart** from the package *rpart* (Therneau, Atkinson, and Ripley 2015) as an implementation of *CART*, **J48** from the package *RWeka* (Hornik, Buchta, and Zeileis 2009) as an implementation of *C4.5*, and **ctree** from the package *party* (Hothorn, Hornik, and Zeileis 2006), which is an implementation of the algorithm *Conditional inference trees*. As for the *Random forest*, we use the implementation **ranger** from the package *ranger* (Wright 2015).

While the random forest learner can be used as-is, the trees can conveniently be combined using mlr's bagging wrapper. The number of trees is set to 100 for both the forest and all bagged tree learners so that this parameter does not influence the results.

```r
# create a random forest learner and three bagged tree learners
lrn1 = makeLearner("classif.ranger", num.trees = 100)
lrn2 = makeBaggingWrapper(makeLearner("classif.rpart"), bw.iters = 100)
lrn3 = makeBaggingWrapper(makeLearner("classif.J48"), bw.iters = 100)
lrn4 = makeBaggingWrapper(makeLearner("classif.ctree"), bw.iters = 100)
```

Query OpenML

Now we search for appropriate tasks on OpenML by querying the server using `listOMLTasks()`, which returns a large data frame:

```r
all.tasks = listOMLTasks()
dim(all.tasks)
```

```
## [1] 8415   20
```

For this study the candidates are filtered to meet the following criteria:
1. Binary classification problem
2. 10-fold cross-validation as resampling procedure 3. No missing values – Random Forest cannot handle them automatically
4. Less than 500 instances – keep evaluation time low
5. $n < p$
6. Predictive accuracy as evaluation measure

Although a data frame can be filtered with the `subset()` function (in R's base package), we strongly recommend the faster and more convenient alternatives provided by either **data.table** (Dowle et al. 2015) or **dplyr** (Wickham 2011).

```r
library(data.table)
tasks = as.data.table(all.tasks)
tasks = tasks[
  task.type == "Supervised Classification" &
  NumberOfClasses == 2 &
  estimation.procedure == "10-fold Crossvalidation" &
  NumberOfMissingValues == 0 &
  NumberOfInstances < 500 &
  NumberOfNumericFeatures < NumberOfInstances &
  evaluation.measures == "predictive_accuracy", ]

nrow(tasks)
```

```
## [1] 196
```

We randomly pick 10 out of the 196 remaining tasks to keep the runtimes reasonable. Furthermore, we have a quick glance at the names of the corresponding data sets.

```r
set.seed(1)
tasks = tasks[sample(nrow(tasks), 10), ]
tasks[, name]
```

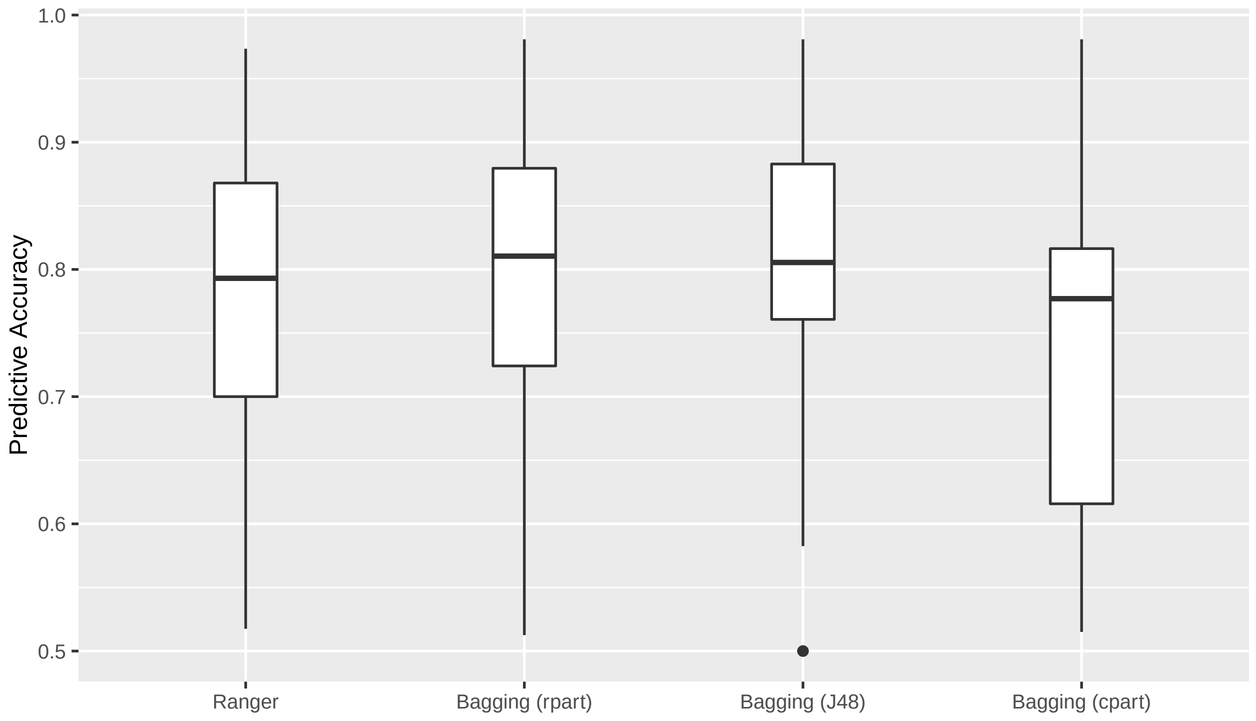
```
##  [1] "fri_c3_100_25"       "pyrim"               "fri_c0_100_25"
##  [4] "jEdit_4.0_4.2"       "fri_c1_250_25"       "ar6"
##  [7] "MegaWatt1"           "chscase_census2"     "chscase_census6"
## [10] "analcatdata_lawsuit"
```

Evaluation

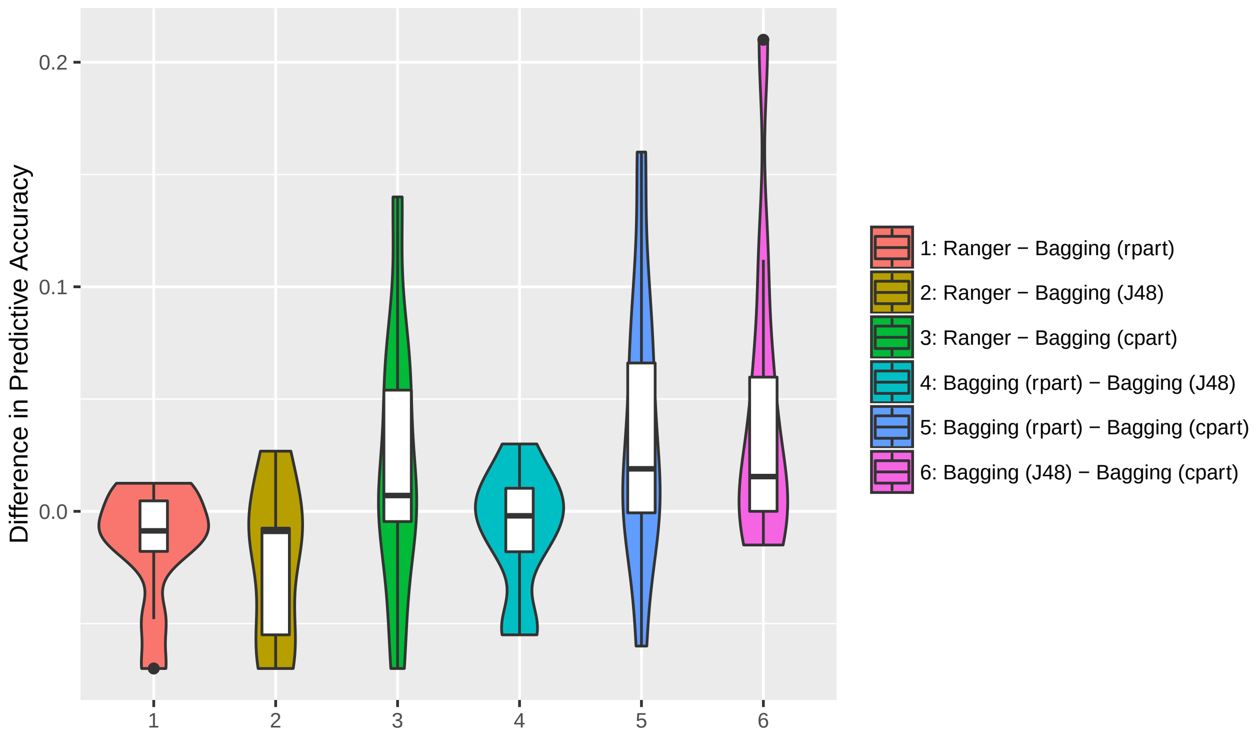
The function `runTaskMlr()` applies an mlr learner on an OpenML data set and returns a benchmark result (`bmr`). Here, we write a short helper function that extracts the aggregated performance measure from this benchmark result for a given task ID and a learner ID determining which of the four learners is run. We then generate a grid of these IDs and map them to the helper function.

```r
runTask = function(task.id, learner.id) {
  res = runTaskMlr(getOMLTask(task.id), learners[[learner.id]])
  getBMRAggrPerformances(res$bmr)[[1]][[1]][1]
}
learners = list(lrn1, lrn2, lrn3, lrn4)
grid = expand.grid(task.id = tasks$task.id, learner.id = 1:4)
res = Map(runTask, task.id = grid$task.id, learner.id = grid$learner.id)
```

In the next figure, the boxplots of the results of the four learners are depicted. We can see a large variance in the predictive accuracies. Apart from that, in average the bagging of cpart trees seems to be the worst of all considered learners. Please note that these results may only indicate tendencies, because we used only a small number of tasks.



To get a closer look at which learner might or might not be better than another, we have a look at the next figure, showing the differences in predictive accuracy per task between each combination of two learners. Besides boxplots, there are also so-called violin plots depicted in the background, that represent the estimated density of the differences' distributions.



References

Dowle, Matt, Arun Srinivasan, Tom Short, Steve Lianoglou with contributions from R Saporta, and E Antonyan. 2015. *Data.table: Extension of Data.frame.* http://CRAN.R-project.org/package=data.table.

Hornik, Kurt, Christian Buchta, and Achim Zeileis. 2009. "Open-Source Machine Learning: R Meets Weka." *Computational Statistics* 24 (2): 225–32. doi:10.1007/s00180-008-0119-7.

Hothorn, Torsten, Kurt Hornik, and Achim Zeileis. 2006. "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics* 15 (3): 651–74.

Therneau, Terry, Beth Atkinson, and Brian Ripley. 2015. *Rpart: Recursive Partitioning and Regression Trees.* http://CRAN.R-project.org/package=rpart.

Wickham, Hadley. 2011. "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software* 40 (1): 1–29. http://www.jstatsoft.org/v40/i01/.

Wright, Marvin N. 2015. *Ranger: A Fast Implementation of Random Forests.* http://CRAN.R-project.org/package=ranger.