How to write a Bot

Example of how to write a simple bot in R

This example explains how to write a bot that runs some experiments with different hyperparameters on some OpenML tasks, upload them on OpenML as runs and get finally a summary table of all the runs that were uploaded to OpenML.

First step: Specification of task, learner and parameter settings

Firstly you have to specify a task, a learner and a corresponding parameter set with which you want to run the experiments with. We randomly draw 10 hyperparameter settings of glmnet here and run it on the task with ID = 3. You can see parameter settings of other algorithms on the bottom of the page.

```r
library(OpenML)
library(mlr)
# Provide dataset(s)
task = getOMLTask(task.id = 3)
# Provide learner(s)
lrn = makeLearner("classif.glmnet")
# Provide parameter setting(s) with ranges (with help of ParamHelpers)
lrn.par.set = makeParamSet(
    makeNumericParam("alpha", lower = 0, upper = 1, default = 1),
    makeNumericVectorParam("lambda", len = 1L, lower = -10, upper = 10,
      default = 0 ,trafo = function(x) 2^x))
# Specify the number of runs
nr.runs = 10
# Draw (random) hyperparameters
par.sets = generateRandomDesign(nr.runs, lrn.par.set, trafo = TRUE)
par.sets = BBmisc::convertDataFrameCols(par.sets, factors.as.char = TRUE)
```

Second Step: Run the experiments and uploading

Run the learner with the randomly drawn different hyperparameters on the dataset and upload them to OpenML

```r
# If you have several learners or tasks, you have to write more for-loops here
for (i in 1:nr.runs) {
  print(i)
  par.set = as.list(par.sets[i,])
  mlr.lrn = setHyperPars(lrn, par.vals = par.set)
  res = runTaskMlr(task, mlr.lrn)
  print(res)
  tags = c("mySimpleBot", "v1")
  uploadOMLRun(res, confirm.upload = FALSE, tags = tags, verbosity = 1)
}
```

Third step: Get final results

Finally you can get the results from OpenML and put them in a summary table.

```r
# Put the results into a table
# get performances
my_runs = listOMLRunEvaluations(tag = "mysimpleBot")
# subset results on some information(s) and measure(s)
my_runs = my_runs[, c("run.id", "task.id", "area.under.roc.curve" )]
# get hyperparameters
parameters = list()
for(i in 1:nrow(my_runs)) {
  run_i = getOMLRun(my_runs$run.id[i])
  parameter_i = data.frame(getOMLRunParList(run_i))
  parameters[[i]] = c(my_runs$run.id[i], as.numeric(parameter_i$value))
}
parameters = do.call(rbind, parameters)
colnames(parameters) = c("run.id", parameter_i$name)
# Put things together
results = merge(my_runs, parameters, by = "run.id")
# Put it in a nice order
results = results[, c(setdiff(names(results), "area.under.roc.curve"), "area.under.roc.curve")]
# Now you can compare the performances of your different hyperparameters
print(results)
```

Get informations about the dataset

If you want to have some informations about the datasets where you benchmarked your models on, you can use `getOMLDataSetQualities`.

```r
# Get some informations about the dataset
data.id = task$input$data.set$desc$id
qualities = getOMLDataSetQualities(data.id = data.id)
qualities[qualities$name %in% c("NumberOfClasses", "NumberOfInstances"), ]
```

Annex

Here we provide some other hyperparameter settings, that you could use to make experiments on OpenML datasets.

```r
# rpart
param.set.rpart = makeParamSet(
  makeNumericParam("cp", lower = 0, upper = 1, default = 0.01),
  makeIntegerParam("maxdepth", lower = 1, upper = 30, default = 30),
  makeIntegerParam("minbucket", lower = 1, upper = 60, default = 1),
  makeIntegerParam("minsplit", lower = 1, upper = 60, default = 20))
# kknn
param.set.kknn = makeParamSet(
  makeIntegerParam("k", lower = 1, upper = 30))
# svm
param.set.svm = makeParamSet(
  makeDiscreteParam("kernel", values = c("linear", "polynomial", "radial")),
  makeNumericParam("cost", lower = -10, upper = 10, trafo = function(x) 2^x),
  makeNumericParam("gamma", lower = -10, upper = 10, trafo = function(x) 2^x, requires = quote(kernel =
  makeIntegerParam("degree", lower = 2, upper = 5, requires = quote(kernel == "polynomial")))
```

```r
# ranger
param.set.ranger = makeParamSet(
  makeIntegerParam("num.trees", lower = 1, upper = 2000),
  makeLogicalParam("replace"),
  makeNumericParam("sample.fraction", lower = 0.1, upper = 1),
  makeNumericParam("mtry", lower = 0, upper = 1),
  makeLogicalParam(id = "respect.unordered.factors"),
  makeNumericParam("min.node.size", lower = 0, upper = 1))
# xgboost
param.set.xgboost = makeParamSet(
  makeIntegerParam("nrounds", lower = 1, upper = 5000),
  makeNumericParam("eta", lower = -10, upper = 0, trafo = function(x) 2^x),
  makeNumericParam("subsample",lower = 0.1, upper = 1),
  makeDiscreteParam("booster", values = c("gbtree", "gblinear")),
  makeIntegerParam("max_depth", lower = 1, upper = 15, requires = quote(booster == "gbtree")),
  makeNumericParam("min_child_weight", lower = 0, upper = 7, requires = quote(booster == "gbtree"), tra
  makeNumericParam("colsample_bytree", lower = 0, upper = 1, requires = quote(booster == "gbtree")),
  makeNumericParam("colsample_bylevel", lower = 0, upper = 1, requires = quote(booster == "gbtree")),
  makeNumericParam("lambda", lower = -10, upper = 10, trafo = function(x) 2^x),
  makeNumericParam("alpha", lower = -10, upper = 10, trafo = function(x) 2^x))
```