Comput Stat (2019) 34:977–991
https://doi.org/10.1007/s00180-017-0742-2

ORIGINAL PAPER

OpenML: An R package to connect to the machine learning platform OpenML

Giuseppe Casalicchio[1] · Jakob Bossek[2] · Michel Lang[3] · Dominik Kirchhoff[4] · Pascal Kerschke[2] · Benjamin Hofner[5] · Heidi Seibold[6] · Joaquin Vanschoren[7] · Bernd Bischl[1]

Received: 23 November 2016 / Accepted: 3 June 2017 / Published online: 19 June 2017

Abstract OpenML is an online machine learning platform where researchers can easily share data, machine learning tasks and experiments as well as organize them online to work and collaborate more efficiently. In this paper, we present an `R` package to interface with the OpenML platform and illustrate its usage in combination with the machine learning `R` package `mlr` (Bischl et al. J Mach Learn Res 17(170):1–5, 2016). We show how the `OpenML` package allows `R` users to easily search, download and upload data sets and machine learning tasks. Furthermore, we also show how to upload results of experiments, share them with others and download results from other users. Beyond ensuring reproducibility of results, the OpenML platform automates much of the drudge work, speeds up research, facilitates collaboration and increases the users' visibility online.

Keywords Databases · Machine learning · R · Reproducible research

✉ Giuseppe Casalicchio
giuseppe.casalicchio@stat.uni-muenchen.de

1 Department of Statistics, Ludwig-Maximilians-University Munich, 80539 Munich, Germany

2 Information Systems and Statistics, University of Münster, 48149 Münster, Germany

3 Department of Statistics, TU Dortmund University, 44227 Dortmund, Germany

4 Dortmund University of Applied Sciences and Arts, 44227 Dortmund, Germany

5 Section of Biostatistics, Paul-Ehrlich-Institut, 63225 Langen, Germany

6 Epidemiology, Biostatistics and Prevention Institute, University of Zurich, 8001 Zurich, Switzerland

7 Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands

1 Introduction

OpenML is an online machine learning platform for sharing and organizing data, machine learning algorithms and experiments (Vanschoren et al. 2013). It is designed to create a frictionless, networked ecosystem (Nielsen 2012), allowing people all over the world to collaborate and build directly on each other's latest ideas, data and results. Key elements of OpenML are data sets, tasks, flows and runs:

- **Data sets** can be shared (under a licence) by uploading them or simply linking to existing data repositories (e.g., mldata.org, figshare.com). For known data formats (e.g., ARFF for tabular data), OpenML will automatically analyze and annotate the data sets with measurable characteristics to support detailed search and further analysis. Data sets can be repeatedly updated or changed and are then automatically versioned.
- **Tasks** can be viewed as containers including a data set and additional information defining what is to be learned. They define which input data are given and which output data should be obtained. For instance, classification tasks will provide the target feature, the evaluation measure (e.g., the area under the curve) and the estimation procedure (e.g., cross-validation splits) as inputs. As output they expect a description of the machine learning algorithm or workflow that was used and, if available, its predictions.
- **Flows** are implementations of single machine learning algorithms or whole workflows that solve a specific task, e.g., a random forest implementation is a flow that can be used to solve a classification or regression task. Ideally, flows are already implemented (or custom) algorithms in existing software that take OpenML tasks as inputs and can automatically read and solve them. They also contain a list (and description) of possible hyperparameters that are available for the algorithm.
- **Runs** are the result of executing flows, optionally with preset hyperparameter values, on tasks and contain all expected outputs and evaluations of these outputs (e.g., the accuracy of predictions). Runs are fully reproducible because they are automatically linked to specific data sets, tasks, flows and hyperparameter settings. They also include the authors of the run and any additional information provided by them, such as runtimes. Similar to data mining challenge platforms (e.g., Kaggle; Carpenter 2011), OpenML evaluates all submitted results (using a range of evaluation measures) and compares them online. The difference, however, is that OpenML is designed for collaboration rather than competition: anyone can browse, immediately build on and extend all shared results.

As an open science platform, OpenML provides important benefits for the science community and beyond.

Benefits for Science: Many sciences have made significant breakthroughs by adopting online tools that help organizing, structuring and analyzing scientific data online (Nielsen 2012). Indeed, any shared idea, question, observation or tool may be noticed by someone who has just the right expertise to spark new ideas, answer open questions, reinterpret observations or reuse data and tools in unexpected new ways. Therefore, sharing research results and collaborating online as a (possibly cross-

disciplinary) team enables scientists to quickly build on and extend the results of others, fostering new discoveries.

Moreover, ever larger studies become feasible as a lot of data are already available. Questions such as "Which hyperparameter is important to tune?", "Which is the best known workflow for analyzing this data set?" or "Which data sets are similar in structure to my own?" can be answered in minutes by reusing prior experiments, instead of spending days setting up and running new experiments (Vanschoren et al. 2012).

Benefits for Scientists: Scientists can also benefit personally from using OpenML. For example, they can *save time*, because OpenML assists in many routine and tedious duties: finding data sets, tasks, flows and prior results, setting up experiments and organizing all experiments for further analysis. Moreover, new experiments are immediately compared to the state of the art without always having to rerun other people's experiments.

Another benefit is that linking one's results to those of others has a large potential for *new discoveries* (Feurer et al. 2015; Post et al. 2016; Probst et al. 2017), leading to more publications and more collaboration with other scientists all over the world. Finally, OpenML can help scientists to *reinforce their reputation* by making their work (published or not) visible to a wide group of people and by showing how often one's data, code and experiments are downloaded or reused in the experiments of others.

Benefits for Society: OpenML also provides a useful learning and working environment for students, citizen scientists and practitioners. Students and citizen scientist can easily explore the state of the art and work together with top minds by contributing their own algorithms and experiments. Teachers can challenge their students by letting them compete on OpenML tasks or by reusing OpenML data in assignments. Finally, machine learning practitioners can explore and reuse the best solutions for specific analysis problems, interact with the scientific community or efficiently try out many possible approaches.

The remainder of this paper is structured as follows. First, we discuss the web services offered by the OpenML server and the website on OpenML.org that allows web access to all shared data and several tools for data organization and sharing. Second, we briefly introduce the `mlr` package (Bischl et al. 2016; Schiffner et al. 2016), which is a machine learning toolbox for `R` (R Core Team 2016) and offers a unified interface to many machine learning algorithms. Third, we discuss and illustrate some important functions of the `OpenML R` package. After that, we illustrate its usage in combination with the `mlr R` package by conducting a short case study. Finally, we conclude with a discussion and an outlook to future developments.

2 The OpenML platform

The OpenML platform consists of several layers of software:

Web API: Any application (or web application), can communicate with the OpenML server through the extensive Web API, an application programming interface (API) that offers a set of calls (e.g., to download/upload data) using representational state transfer (REST) which is a simple, lightweight communication mechanism based on standard HTTP requests. Data sets, tasks, flows and runs can be created, read, updated, deleted, searched and tagged through simple HTTP calls. An overview of calls is available on http://www.openml.org/api_docs.

Website: OpenML.org is a website offering easy browsing, organization and sharing of all data, code and experiments. It allows users to easily search and browse all shared data sets, tasks, flows and runs, as well as to compare and visualize all combined results. It provides an easy way to check and manage your experiments anywhere, anytime and discuss them with others online. See Fig. 1 for a few screenshots of the OpenML website.

Programming Interfaces: OpenML also offers interfaces in multiple programming languages, such as the `R` interface presented here, which hides the API calls and allow scientists to interact with the server using language-specific functions. As we demonstrate below, the `OpenML R` package allows `R` users to search and download data sets and upload the results of machine learning experiments in just a few lines of code. Other interfaces exist for `Python`, `Java` and `C# (.NET)`. For tools that usually operate through a graphical interface, such as WEKA (Hall et al. 2009), MOA (Bifet et al. 2010) and RapidMiner (van Rijn et al. 2013), plug-ins exist that integrate OpenML sharing facilities.

OpenML is organized as an open source project, hosted on GitHub (https://github.com/openml) and is free to use under the CC-BY licence. When uploading new data sets and code, users can select under which licence they wish to share the data, OpenML will then state licences and citation requests online and in descriptions downloaded from the Web API.

OpenML has an active developer community and everyone is welcome to help extend it or post new suggestions through the website or through GitHub. Currently, there are close to 1,700,000 runs on about 20,000 data sets and 3500 unique flows available on the OpenML platform. While still in beta development, it has over 1400 registered users, over 1800 frequent visitors and the website is visited by around 200 unique visitors every day, from all over the world. It currently has server-side support for classification, regression, clustering, data stream classification, learning curve analysis, survival analysis and machine learning challenges for classroom use.

3 The mlr R package

The `mlr` package (Bischl et al. 2016; Schiffner et al. 2016) offers a clean, easy-to-use and flexible domain-specific language for machine learning experiments in `R`. An object-oriented interface is adopted to unify the definition of machine learning tasks, setup of learning algorithms, training of models, predicting and evaluating

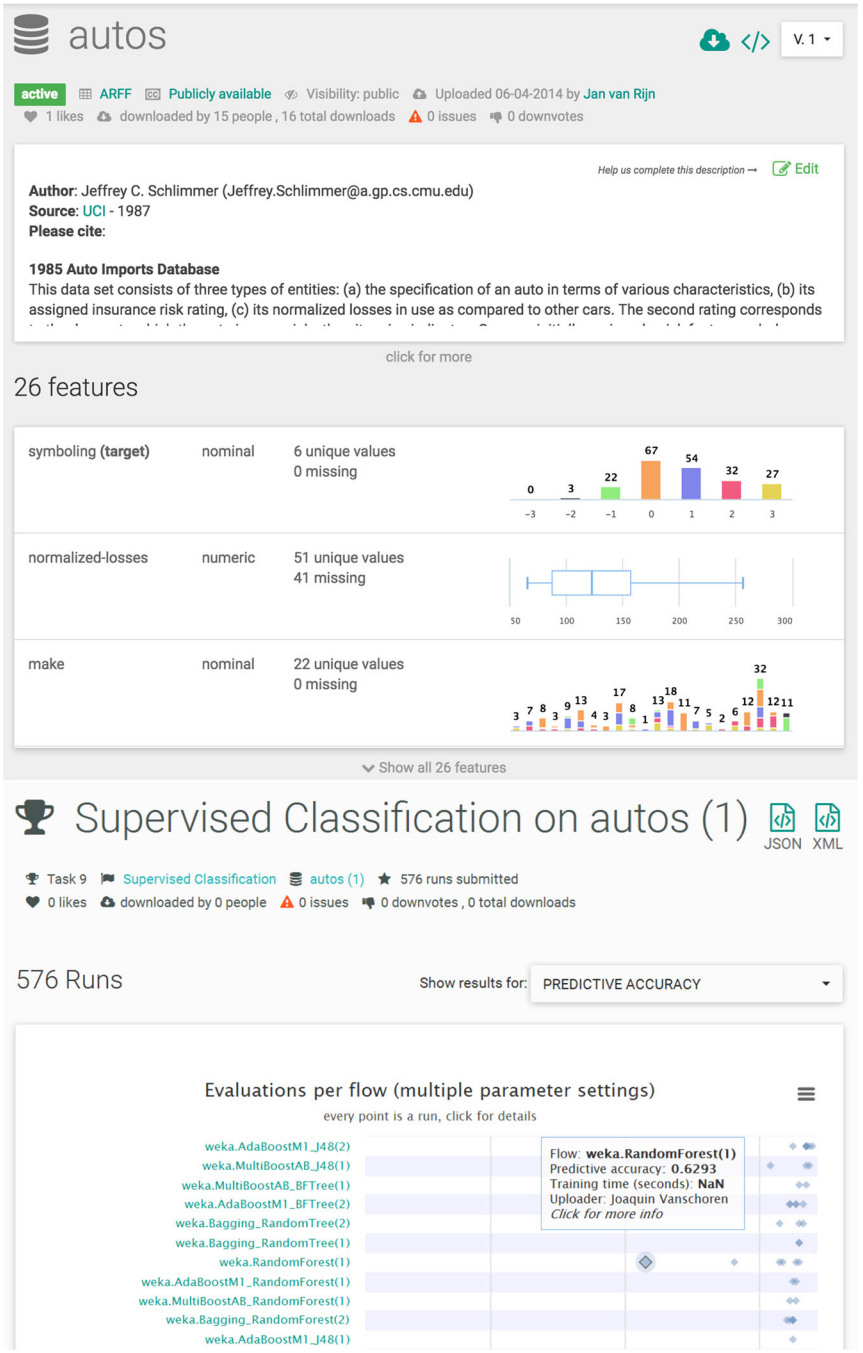

Fig. 1 Screenshots of the OpenML website. The *top part* shows the data set 'autos', with wiki description and descriptive overview of the data features. The *bottom part* shows a classification task, with an overview of the best submitted flows with respect to the predictive accuracy as performance measure. Every *dot* here is a single run (further to the right is better)

the algorithm's performance. This unified interface hides the actual implementations of the underlying learning algorithms. Replacing one learning algorithm with another becomes as easy as changing a string. Currently, `mlr` has built-in support for classification, regression, multilabel classification, clustering and survival analysis and includes in total 160 modelling techniques. A complete list of the integrated learners and how to integrate own learners, as well as further information on the `mlr` package can be found in the corresponding tutorial (http://mlr-org.github.io/mlr-tutorial/). A plethora of further functionality is implemented in `mlr`, e.g., assessment of generalization performance, comparison of different algorithms in a scientifically rigorous way, feature selection and algorithms for hyperparameter tuning, including Iterated F-Racing (Lang et al. 2015) and Bayesian optimization with the package `mlrMBO` (Bischl et al. 2017). On top of that, `mlr` offers a wrapper mechanism, which allows to extend learners through pre-train, post-train, pre-predict and post-predict hooks. A wrapper extends the current learner with added functionality and expands the hyperparameter set of the learner with additional hyperparameters provided by the wrapper. Currently, many wrappers are available, e.g., missing value imputation, class imbalance correction, feature selection, tuning, bagging and stacking, as well as a wrapper for user-defined data pre-processing. Wrappers can be nested in other wrappers, which can be used to create even more complex workflows. The package also supports parallelization on different levels based on different parallelization backends (local multicore, socket, MPI) with the package `parallelMap` (Bischl and Lang 2015) or on managed high-performance systems via the package `batchtools` (Lang et al. 2017). Furthermore, visualization methods for research and teaching are also supplied.

The `OpenML` package makes use of `mlr` as a supporting package. It offers methods to automatically run `mlr` learners (flows) on OpenML tasks while hiding all of the necessary structural transformations (see Sect. 4.4).

4 The OpenML R package

The `OpenML R` package Casalicchio et al. (2017) is an interface to interact with the OpenML server directly from within `R`. Users can retrieve data sets, tasks, flows and runs from the server and also create and upload their own. This section details how to install and configure the package and demonstrates its most important functionalities.

4.1 Installation and configuration

To interact with the OpenML server, users need to authenticate using an *API key*, a secret string of characters that uniquely identifies the user. It is generated and shown on users' profile page after they register on the website http://www.openml.org. For demonstration purposes, we will use a public read-only API key that only allows to retrieve information from the server and should be replaced with the user's personal API key to be able to use all features. The `R` package can be easily installed and configured as follows:

```
install.packages("OpenML")
library("OpenML")
saveOMLConfig(apikey = "c1994bdb7ecb3c6f3c8f3b35f4b47f1f")
```

The `saveOMLConfig` function creates a `config` file, which is always located in a folder called `.openml` within the user's home directory. This file stores the user's API key and other configuration settings, which can always be changed manually or through the `saveOMLConfig` function. Alternatively, the `setOMLConfig` function allows to set the API key and the other entries *temporarily*, i.e., only for the current `R` session.

4.2 Listing information

In this section, we show how to list the available OpenML data sets, tasks, flows and runs using listing functions that always return a `data.frame` containing the queried information. Each data set, task, flow and run has a unique ID, which can be used to access it directly.

Listing Data Sets and Tasks: A list of all data sets and tasks that are available on the OpenML server can be obtained using the `listOMLDataSets` and `listOMLTasks` function, respectively. Each entry provides information such as the ID, the name and basic characteristics (e.g., number of features, number of observations, classes, missing values) of the corresponding data set. In addition, the list of tasks contains information about the task type (e.g., `"Supervised Classification"`), the evaluation measure (e.g., `"Predictive Accuracy"`) and the estimation procedure (e.g., `"10-fold Crossvalidation"`) used to estimate model performance. Note that multiple tasks can be defined for a specific data set, for example, the same data set can be used for multiple task types (e.g. classification and regression tasks) as well as for tasks differing in their estimation procedure, evaluation measure or target value.

To find data sets or tasks that meet specific requirements, one can supply arguments to the listing functions. In the example below, we list all supervised classification tasks based on data sets having two classes for the target feature, between 500 and 999 instances, at most 100 features and no missing values:

```
tasks = listOMLTasks(task.type = "Supervised Classification",
  number.of.classes = 2, number.of.instances = c(500, 999),
  number.of.features = c(1, 100), number.of.missing.values = 0)
tasks[1:2, c("task.id", "name", "number.of.instances", "number.of.features")]
##   task.id        name number.of.instances number.of.features
## 1      37    diabetes                768                 9
## 2      49 tic-tac-toe                958                10
```

Listing Flows and Runs: When using the `mlr` package, flows are basically learners from `mlr`, which, as stated previously, can also be a more complex workflow when different `mlr` wrappers are nested. Custom flows can be created by integrating custom machine learning algorithms and wrappers into `mlr`. The list of all available flows on OpenML can be downloaded using the `listOMLFlows` function. Each entry contains

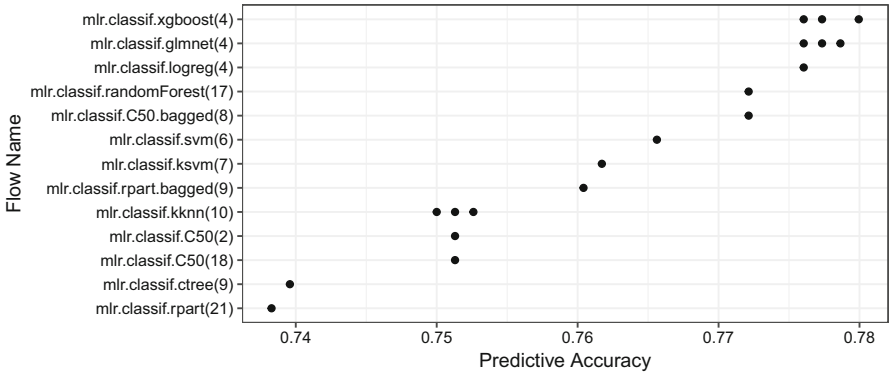

Fig. 2 The predictive accuracy of some `mlr` flows on task 37. The numbers in *brackets* refer to the version of the flow. Multiple *dots* for the same flow refer to runs with different hyperparameter values for that flow

information such as its ID, its name, its version and the user who first uploaded the flow to the server. Note that the list of flows will not only contain flows created with `R`, but also flows from other machine learning toolkits, such as WEKA (Hall et al. 2009), MOA (Bifet et al. 2010) and scikit-learn (Pedregosa et al. 2011), which can be recognized by the name of the flow.

When a flow, along with a specific setup (e.g., specific hyperparameter values), is applied to a task, it creates a run. The `listOMLRuns` function lists all runs that, for example, refer to a specific `task.id` or `flow.id`. To list these evaluations as well, the `listOMLRunEvaluations` function can be used. In Fig. 2, we used `ggplot2` (Wickham 2009) to visualize the predictive accuracy of runs, for which only flows created with `mlr` were applied to the task with ID `37`:

```
res = listOMLRunEvaluations(task.id = 37, tag = "openml_r_paper")
res$flow.name = reorder(res$flow.name, res$predictive.accuracy)

library("ggplot2")
ggplot(res, aes(x = predictive.accuracy, y = flow.name)) +
  geom_point() + xlab("Predictive Accuracy") + ylab("Flow Name")
```

4.3 Downloading OpenML objects

Most of the listing functions described in the previous section will list entities by their OpenML IDs, e.g., the `task.id` for tasks, the `flow.id` for flows and the `run.id` for runs. In this section, we show how these IDs can be used to download a certain data set, task, flow or run from the OpenML server. All downloaded data sets, tasks, flows and runs will be stored in the `cachedir` directory, which will be in the `.openml` folder by default but can also be specified in the configuration file (see Sect. 4.1). Before downloading an OpenML object, the cache directory will be checked if that object is already available in the cache. If so, no internet connection is necessary and the requested object is retrieved from the cache.

Downloading Data Sets and Tasks: The `getOMLDataSet` function returns an `S3`-object of class `OMLDataSet` that contains the data set as a `data.frame` in a `$data` slot, in addition to some pieces of meta-information:

```
ds = getOMLDataSet(data.id = 15)
ds
##
## Data Set "breast-w" :: (Version = 1, OpenML ID = 15)
##   Default Target Attribute: Class
```

To retrieve tasks, the `getOMLTask` function can be used with their corresponding task ID. Note that the ID of a downloaded task is not equal to the ID of the data set. Each task is returned as an `S3`-object of class `OMLTask` and contains the `OMLDataSet` object as well as the predefined estimation procedure, evaluation measure and the target feature in an additional `$input` slot. Further technical information can be found in the package's help page.

Downloading Flows and Runs: The `getOMLFlow` function downloads all information of the flow, such as the name, all necessary dependencies and all available hyperparameters that can be set. If the flow was created in `R`, it can be converted into an `mlr` learner using the `convertOMLFlowToMlr` function:

```
mlr.lrn = convertOMLFlowToMlr(getOMLFlow(4782))
mlr.lrn
## Learner classif.randomForest from package randomForest
## Type: classif
## Name: Random Forest; Short name: rf
## Class: classif.randomForest
## Properties: twoclass,multiclass,numerics,factors,ordered,prob,class.weights
## Predict-Type: response
## Hyperparameters:
```

This allows users to apply the downloaded learner to other tasks or to modify the learner using functions from `mlr` and produce new runs.

The `getOMLRun` function downloads a single run and returns an `OMLRun` object containing all information that are connected to this run, such as the ID of the task and the ID of the flow:

```
run = getOMLRun(run.id = 1816245)
run
##
## OpenML Run 1816245 :: (Task ID = 42, Flow ID = 4782)
##  User ID  : 348
##  Tags     : study_30
##  Learner  : mlr.classif.randomForest(17)
##  Task type: Supervised Classification
```

The most important information for reproducibility, next to the exact data set and flow version, are the hyperparameter and seed settings that were used to create this run. This information is contained in the `OMLRun` object and can be extracted via `getOMLRunParList(run)` and `getOMLSeedParList(run)`, respectively.

If the run solves a supervised regression or classification task, the corresponding predictions can be accessed via `run$predictions` and the evaluation measures computed by the server via `run$output.data$evaluations`.

4.4 Creating runs

The easiest way to create a run is to define a learner, optionally with a preset hyperparameter value, using the `mlr` package. Each `mlr` learner can then be applied to a specific `OMLTask` object using the function `runTaskMlr`. This will create an `OMLMlrRun` object, for which the results can be uploaded to the OpenML server as described in the next section. For example, a random forest from the `randomForest` `R` package (Liaw and Wiener 2002) can be instantiated using the `makeLearner` function from `mlr` and can be applied to a classification task via:

```
lrn = makeLearner("classif.randomForest", mtry = 2)
task = getOMLTask(task.id = 37)
run.mlr = runTaskMlr(task, lrn)
```

To run a previously downloaded OpenML flow, one can use the `runTaskFlow` function, optionally with a list of hyperparameters:

```
flow = getOMLFlow(4782)
run.flow = runTaskFlow(task, flow, par.list = list(mtry = 2))
```

To display benchmarking results, one can use the `convertOMLMlrRunToBMR` function to convert one or more `OMLMlrRun` objects to a single `BenchmarkResult` object from the `mlr` package so that several powerful plotting functions (see http://mlr-org.github.io/mlr-tutorial/release/html/benchmark_experiments for examples) from `mlr` can be applied to that object (see, e.g., Fig. 3).

4.5 Uploading and tagging

Uploading OpenML Objects: It is also possible to upload data sets, flows and runs to the OpenML server to share and organize experiments and results online. Data sets, for example, are uploaded with the `uploadOMLDataSet` function. OpenML will *activate* the data set if it passes all checks, meaning that it will be returned in listing calls. Creating tasks from data sets is currently only possible through the website, see http://www.openml.org/new/task.

`OMLFlow` objects can be uploaded to the server with the `uploadOMLFlow` function and are automatically versioned by the server: when a learner is uploaded carrying a different `R` or package version, a new version number and `flow.id` is assigned. If the same flow has already been uploaded to the server, a message that the flow already exists is displayed and the associated `flow.id` is returned. Otherwise, the flow is uploaded and a new `flow.id` is assigned to it:

```
lrn = makeLearner("classif.randomForest")
flow.id = uploadOMLFlow(lrn)
```

A run created with the `runTaskMlr` or the `runTaskFlow` function can be uploaded to the OpenML server using the `uploadOMLRun` function. The server will then automatically compute several evaluation measures for this run, which can be retrieved using the `listOMLRunEvaluations` function as described previously.

Tagging and Untagging OpenML Objects: The `tagOMLObject` function is able to tag data sets, tasks, flows and runs with a user-defined string, so that finding OpenML objects with a specific tag becomes easier. For example, the task with ID 1 can be tagged as follows:

```
tagOMLObject(id = 1, object = "task", tags = "test-tagging")
```

To retrieve a list of objects with a given tag, the `tag` argument of the listing functions can be used (e.g., `listOMLTasks(tag = "test-tagging")`). The listing functions for data sets, tasks, flows and runs also show the tags that were already assigned, for example, we already tagged data sets from UCI (Asuncion and Newman 2007) with the string `"uci"` so that they can be queried using `listOMLDataSets(tag = "uci")`. In order to remove one or more tags from an `OpenML` object, the `untagOMLObject` function can be used, however, only self-created tags can be removed, e.g.:

```
untagOMLObject(id = 1, object = "task", tags = "test-tagging")
```

4.6 Further features

Besides the aforementioned functionalities, the `OpenML` package allows to fill up the cache directory by downloading multiple objects at once (using the `populateOMLCache` function), to remove all files from the cache directory (using `clearOMLCache`), to get the current status of cached data sets (using `getCachedOMLDataSetStatus`), to delete OpenML objects created by the uploader (using `deleteOMLObject`), to list all estimation procedures (using `listOMLEstimationProcedures`) as well as all available evaluation measures (using `listOMLEvaluationMeasures`) and to get more detailed information on data sets (using `getOMLDataSetQualities`).

5 Case study

In this section, we illustrate the usage of OpenML by performing a small comparison study between a random forest, bagged trees and single classification trees. We first create the respective binary classification learners using `mlr`, then query OpenML for suitable tasks, apply the learners to the tasks and finally evaluate the results.

5.1 Creating learners

We choose three implementations of different tree algorithms, namely the *CART* algorithm implemented in the `rpart` package (Therneau et al. 2015), the *C5.0* algorithm from the package `C50` (Kuhn et al. 2015) and the *conditional inference trees* implemented in the `ctree` function from the package `party` (Hothorn et al. 2006). For the *random forest*, we use the implementation from the package `randomForest` (Liaw and Wiener 2002). The bagged trees can conveniently be created using `mlr`'s bagging wrapper. Note that we do not use bagging for the `ctree` algorithm due to large

memory requirements. For the random forest and all bagged tree learners, the number of trees is set to 50. We create a list that contains the random forest, the two bagged trees and the three tree algorithms:

```
lrn.list = list(
  makeLearner("classif.randomForest", ntree = 50),
  makeBaggingWrapper(makeLearner("classif.rpart"), bw.iters = 50),
  makeBaggingWrapper(makeLearner("classif.C50"), bw.iters = 50),
  makeLearner("classif.rpart"),
  makeLearner("classif.C50"),
  makeLearner("classif.ctree")
)
```

5.2 Querying OpenML

For this study, we consider only binary classification tasks that use smaller data sets from UCI (Asuncion and Newman 2007), e.g., between 100 and 999 observations, have no missing values and use 10-fold cross-validation for validation:

```
tasks = listOMLTasks(data.tag = "uci",
  task.type = "Supervised Classification", number.of.classes = 2,
  number.of.missing.values = 0, number.of.instances = c(100, 999),
  estimation.procedure = "10-fold Crossvalidation")
```

Table 1 shows the resulting tasks of the query, which will be used for the further analysis.

5.3 Evaluating results

We now apply all learners from `lrn.list` to the selected tasks using the `runTaskMlr` function and use the `convertOMLMlrRunToBMR` function to create a single `BenchmarkResult` object containing the results of all experiments. This allows using, for example, the `plotBMRBoxplots` function from `mlr` to visualize the experiment results (see Fig. 3):

Table 1 Overview of OpenML tasks that will be used in the study

Task id	Name	Number of instances	Number of features
37	Diabetes	768	9
39	Sonar	208	61
42	Haberman	306	4
49	Tic-tac-toe	958	10
52	Heart-statlog	270	14
57	Ionosphere	351	35

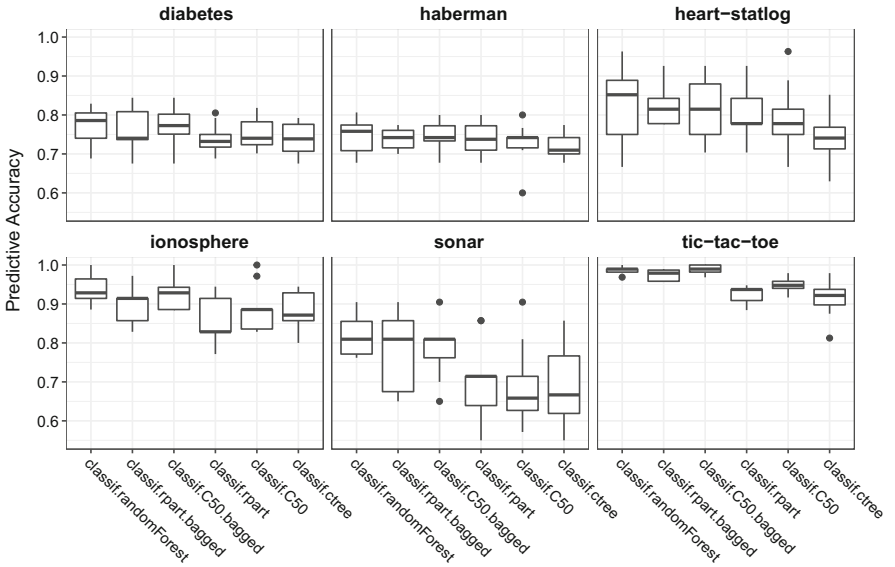

Fig. 3 Cross-validated predictive accuracy per learner and task. Each *boxplot* contains 10 values for one complete cross-validation

```
grid = expand.grid(task.id = tasks$task.id, lrn.ind = seq_along(lrn.list))
runs = lapply(seq_row(grid), function(i) {
  task = getOMLTask(grid$task.id[i])
  ind = grid$lrn.ind[i]
  runTaskMlr(task, lrn.list[[ind]])
})
bmr = do.call(convertOMLMlrRunToBMR, runs)
plotBMRBoxplots(bmr, pretty.names = FALSE)
```

We can upload and tag the runs, e.g., with the string `"study_30"` to facilitate finding and listing the results of the runs using this tag:

```
lapply(runs, uploadOMLRun, tags = "study_30")
```

The server will then compute all possible measures, which takes some time depending on the number of runs. The results can then be listed using the `listOMLRunEvaluations` function and can be visualized using the `ggplot2` package:

```
evals = listOMLRunEvaluations(tag = "study_30")
evals$learner.name = as.factor(evals$learner.name)
evals$task.id = as.factor(evals$task.id)

library("ggplot2")
ggplot(evals, aes(x = data.name, y = predictive.accuracy, colour = learner.name,
  group = learner.name, linetype = learner.name, shape = learner.name)) +
  geom_point() + geom_line() + ylab("Predictive Accuracy") + xlab("Data Set") +
  theme(axis.text.x = element_text(angle = -45, hjust = 0))
```

Figure 4 shows the cross-validated predictive accuracies of our six learners on the considered tasks. Here, the random forest produced the best predictions, except on the

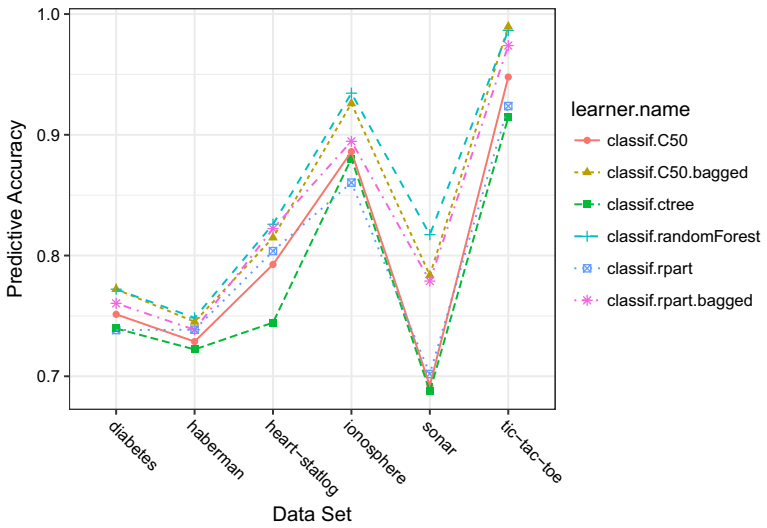

Fig. 4 Results of the produced runs. Each *point* represents the averaged predictive accuracy over all cross-validation iterations generated by running a particular learner on the respective task

tic-tac-toe data set, where the bagged C50 trees achieved a slightly better result. In general, the two bagged trees performed marginally worse than the random forest and better than the single tree learners.

6 Conclusion and outlook

OpenML is an online platform for open machine learning that is aimed at connecting researchers who deal with any part of the machine learning workflow. The OpenML platform automates the sharing of machine learning tasks and experiments through the tools that scientists are already using, such as `R`. The `OpenML` package introduced in this paper makes it easy to share and reuse data sets, tasks, flows and runs directly from the current `R` session without the need of using other programming environments or the web interface.

Current work is being done on implementing the possibility to connect to OpenML via browser notebooks (https://github.com/everware) and running analysis directly on online servers without the need of having `R` or any other software installed locally. In the future, it will also be possible that users can specify with whom they want to share, e.g., data sets.

References

Asuncion A, Newman DJ (2007) UCI Machine Learning Repository. University of California, School of Information and Computer Science

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: Massive online analysis. J Mach Learn Res 11:1601–1604 http://www.jmlr.org/papers/v11/bifet10a.html

Bischl B, Lang M (2015) parallelMap: Unified Interface to Parallelization Back-Ends. https://CRAN.R-project.org/package=parallelMap, r package version 1.3
Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016) mlr: Machine learning in R. J Mach Learn Res 17(170):1–5, http://jmlr.org/papers/v17/15-066.html
Bischl B, Richter J, Bossek J, Horn D, Thomas J, Lang M (2017) mlrmbo: A modular framework for model-based optimization of expensive black-box functions. arXiv preprint arXiv:1703.03373
Carpenter J (2011) May the best analyst win. Science 331(6018):698–699
Casalicchio G, Bischl B, Kirchhoff D, Lang M, Hofner B, Bossek J, Kerschke P, Vanschoren J (2017) OpenML: Exploring machine learning better, together. https://CRAN.R-project.org/package=OpenML, R package version 1.3
Feurer M, Springenberg JT, Hutter F (2015) Initializing bayesian hyperparameter optimization via meta-learning. In: AAAI, pp 1128–1135
Hall MA, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explor Newslett 11(1):10–18, http://www.cs.waikato.ac.nz/ml/weka/
Hothorn T, Hornik K, Zeileis A (2006) Unbiased recursive partitioning: a conditional inference framework. J Comput Gr Stat 15(3):651–674
Kuhn M, Weston S, Coulter N, Culp M (2015) C50: C5.0 decision trees and rule-based models. https://CRAN.R-project.org/package=C50, R package version 0.1.0-24, C code for C5.0 by R. Quinlan
Lang M, Kotthaus H, Marwedel P, Weihs C, Rahnenführer J, Bischl B (2015) Automatic model selection for high-dimensional survival analysis. J Stat Comput Simul 85(1):62–76
Lang M, Bischl B, Surmann D (2017) batchtools: Tools for r to work on batch systems. J Open Source Softw 2(10), https://doi.org/10.21105%2Fjoss.00135
Liaw A, Wiener M (2002) Classification and regression by randomForest. R News 2(3):18–22, http://CRAN.R-project.org/doc/Rnews/
Nielsen M(2012) Reinventing discovery: the new era of networked science. Princeton University Press, http://www.jstor.org/stable/j.ctt7s4vx
Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830, http://scikit-learn.org/
Post MJ, van der Putten P, van Rijn JN (2016) Does feature selection improve classification? a large scale experiment in OpenML. In: International Symposium on Intelligent Data Analysis, Springer, pp 158–170
Probst P, Au Q, Casalicchio G, Stachl C, Bischl B (2017) Multilabel classification with R package mlr. arXiv preprint arXiv:1703.08991
R Core Team (2016) R: A language and environment for statistical computing. R Foundation for statistical computing, Vienna, Austria, https://www.R-project.org/
Schiffner J, Bischl B, Lang M, Richter J, Jones ZM, Probst P, Pfisterer F, Gallo M, Kirchhoff D, Kühn T, Thomas J, Kotthoff L (2016) mlr Tutorial. arXiv preprint arXiv:1609.06146
Therneau T, Atkinson B, Ripley B (2015) rpart: Recursive Partitioning and Regression Trees. http://CRAN.R-project.org/package=rpart, R package version 4.1-10
van Rijn JN, Umaashankar V, Fischer S, Bischl B, Torgo L, Gao B, Winter P, Wiswedel B, Berthold MR, Vanschoren J (2013) A RapidMiner Extension for Open Machine Learning. In: Proceedings of the 4th RapidMiner Community Meeting and Conference (RCOMM 2013), pp 59–70
Vanschoren J, Blockeel H, Pfahringer B, Holmes G (2012) Experiment Databases. A new way to share, organize and learn from experiments. Mach Learn 87(2):127–158
Vanschoren J, van Rijn JN, Bischl B, Torgo L (2013) OpenML: networked science in machine learning. SIGKDD Explor 15(2):49–60
Wickham H (2009) ggplot2: Elegant Graphics for Data Analysis. Springer, New York, NY, USA, http://ggplot2.org