A RapidMiner extension for Open Machine Learning

Jan N. van Rijn[1], Venkatesh Umaashankar[2], Simon Fischer[2], Bernd Bischl[3], Luis Torgo[4], Bo Gao[5], Patrick Winter[6], Bernd Wiswedel[6], Michael R. Berthold[7] and Joaquin Vanschoren[1]

[1]Leiden University, {jvrijn,joaquin}@liacs.nl
[2]Rapid-I GmbH, {venkatesh,fischer}@rapid-i.com
[3]TU Dortmund, Dortmund, Germany,
bischl@statistik.tu-dortmund.de
[4]University of Porto, Porto, Portugal, ltorgo@inescporto.pt
[5]KU Leuven, Leuven, Belgium, bo.gao@cs.kuleuven.be
[6]KNIME.com AG,
{patrick.winter,Bernd.Wiswedel}@knime.com
[7]University of Konstanz, Konstanz, Germany,
Michael.Berthold@uni-konstanz.de

Abstract

We present a RapidMiner extension for OpenML, an open science platform for sharing machine learning datasets, algorithms and experiments. In order to share machine learning experiments as easily as possible, it is being integrated into various popular data mining and machine learning tools, including RapidMiner. Through this plugin, data can be downloaded, and workflows and results uploaded to the OpenML website, where they can be searched, aggregated and reused.

1 Introduction

In this paper we present a RapidMiner extension for OpenML[1], an open science platform for machine learning. It allows researchers to submit datasets, algorithms, workflows, experiments and their results to a single platform. OpenML

[1] http://openml.org/, beta

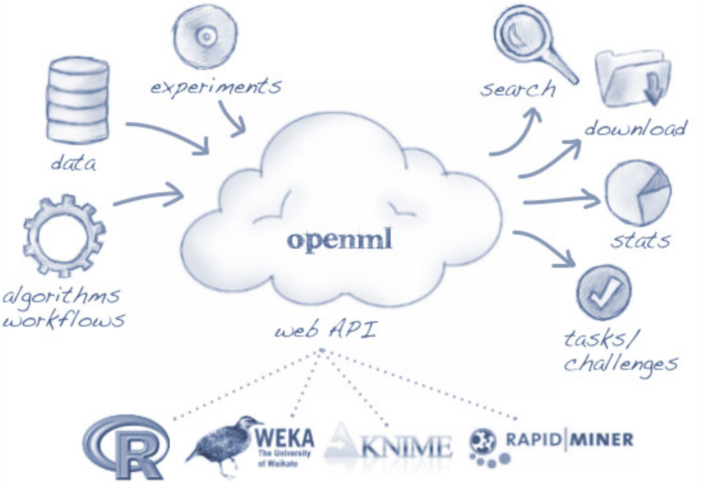


Figure 1: Components of OpenML.

automatically organizes all content in a database, where it is freely available to everyone and searchable through its website. Above all, OpenML aims to facilitate an open scientific culture, which in turn can tremendously speed up progress [5]. First, by publishing detailed results, other scientists can clearly interpret, and even verify and reproduce certain findings, so that they can confidently build upon prior work [4]. Furthermore, by integrating results from many studies, researchers can conduct much larger studies. Algorithms and workflows can immediately be compared over many different datasets and a wide range of varying parameter settings. Finally, many machine learning questions won't require the set up of new experiments. These can be answered on the fly by searching and combining results from earlier studies.

The key components of OpenML are shown in Figure 1. Central are the OpenML API and database, which contains all details and meta-data about all shared datasets, implementations and experiments. New content can be sent to OpenML by means of a RESTful API. This API is also being integrated into a number of popular data mining and machine learning tools, i.e., Weka, R, KNIME and RapidMiner, so that content can be shared automatically. OpenML also provides various search interfaces, so that these entities can later be retrieved, e.g., through a web interface, textual search engine or SQL interface. The latter enables users to directly query the database by means of SQL statements.

In Section 2 an overview of related work is provided. In Section 3, we discuss how experiments are defined in OpenML. Section 4 provides an overview of the most important concepts in the database. In Section 5 we describe the web API, allowing integration into various tools. Section 6 describes how we support the sharing of experiments, and how it is being integrated into RapidMiner. Section 7 details the search interfaces of OpenML, and Section 8 concludes.

2 Related Work

OpenML builds upon previous work on experiment databases [7], which introduced the idea of sharing machine learning experiments in databases for in-depth analysis of learning algorithms. The most notable enhancement of OpenML is the introduction of a web API to allow integration with various machine learning tools, and a more clear definition of experiments through *tasks* (see Section 3).

Kaggle[2] is a platform which hosts machine learning challenges. In some sense, these challenges are similar to OpenML tasks: users are provided with data and instructions, and challenged to build a predictive model on this. However, as Kaggle is a competition platform, it does not support collaboration: people are not allowed to see each other's algorithms or results. OpenML, however, is an open science platform, allowing researchers complete insight into each other's work.

There also exist several platforms for sharing algorithms, workflows and datasets, such as myExperiment [1, 3] and MLData.[3] However, these platforms were not designed to collect and organise large amounts of experimental results over many algorithms and datasets, nor allow such detailed analysis of learning algorithms and workflows afterwards. On the other hand, we do aim to fully integrate these platforms with OpenML, so that datasets and algorithms can be easily transferred between them.

Finally, MLComp[4] is a service that offers to run your algorithms on a range of datasets (or vice versa) on their servers. This has the great benefit that runtimes can be compared more easily. This is not strictly possible in OpenML, because experiments are typically run on the user's machines. However, OpenML does allow you to rerun the exact same experiments on different hardware, which is necessary anyway since hardware will change over time. Moreover, researchers do not need to adapt the way they do their research: they can run their algorithms in their own environments. OpenML also allows users to define different types of experiments beyond the traditional benchmarking runs, and allows more flexible search and query capabilities beyond direct algorithm comparisons.

[2] http://www.kaggle.com/

[3] http://www.mldata.org/

[4] http://www.mlcomp.org/

3 Tasks

In order to make experiments from different researchers comparable, OpenML fully defines experiments in *tasks*. A task is a well-defined problem to be solved by a machine learning algorithm or workflow. For each task, the inputs are provided and the expected output is defined. An attempt to solve a task is called a *run*. Currently, tasks of the type *Supervised Classification* and *Supervised Regression* are supported, but OpenML is designed in such a way that it can be extended with other task types. A typical task would be: *Predict target variable X on dataset Y with a maximized score for evaluation metric Z.* Usually, when a user wants to submit new results to the server, he searches for appropriate tasks, and runs his algorithm on these. The results from these runs will be uploaded to the server, along with information about the algorithm, its version and the parameter settings. This process is explained in detail in Section 5.

```
<oml:task xmlns:oml="http://open-ml.org/openml">
  <oml:task_id>2</oml:task_id>
  <oml:task_type>Supervised Classification</oml:task_type>
  <oml:input name="source_data">
    <oml:data_set>
      <oml:data_set_id>61</oml:data_set_id>
      <oml:target_feature>class</oml:target_feature>
    </oml:data_set>
  </oml:input>
  <oml:input name="estimation_procedure">
    <oml:estimation_procedure>
      <oml:type>cross_validation</oml:type>
      <oml:data_splits_url>
        http://www.openml.org/data/splits/iris_splits.arff
      </oml:data_splits_url>
      <oml:parameter name="number_folds">10</oml:parameter>
      <oml:parameter name="number_repeats">10</oml:parameter>
      <oml:parameter name="stratified_sampling">true</oml:parameter>
    </oml:estimation_procedure>
  </oml:input>
  <oml:input name="evaluation_measures">
    <oml:evaluation_measures>
      <oml:evaluation_measure>predictive_accuracy</oml:evaluation_measure>
    </oml:evaluation_measures>
  </oml:input>
  <oml:output name="predictions">
    <oml:predictions>
      <oml:format>ARFF</oml:format>
      <oml:feature name="confidence.classname" type="numeric" />
      <oml:feature name="fold" type="integer" />
      <oml:feature name="prediction" type="string" />
      <oml:feature name="repeat" type="integer" />
      <oml:feature name="row_id" type="integer" />
    </oml:predictions>
  </oml:output>
</oml:task>
```

Figure 2: XML representation of a task.

Figure 2 shows an example of a Supervised Classification task definition. It provides all information necessary for executing it, such as a URL to download the input dataset and an *estimation procedure*. The estimation procedure describes how the algorithms that are run on this task are being evaluated, e.g., using cross validation, a holdout set or leave-one-out. For every run performed on a certain task, this is done using the same data splits. An ARFF file containing these splits is provided. Also, a set of *evaluation measures* to optimise on is provided. An ARFF file containing the predictions (and confidences per class) is expected as the result.

4 Database

One of the key aspects of OpenML is the central database, containing details about all experiments. A partial schema of the database is provided in Figure 3. In the database schema, the concept of *inheritance* is used: some tables shown do not exist, but describe what fields should be contained by tables inheriting from them, i.e., `data` and `setup`. We call these tables *interface tables*. Also, all tables inheriting from the same interface table share a primary key.

OpenML considers algorithms as conceptual entities, an algorithm itself can not be used to execute a task. Instead, an algorithm can be implemented, resulting in an *implementation*. In this way we also support versioning. For example, when an implementation containing a bug is used, this will potentially yield suboptimal results. This changes whenever a new version is released. Thus, we use the table `implementation`, where the primary key is `fullName`, an aggregation of its name (field: `implementation`) and its version (field: `version`). More specifically, an implementation can typically be run with different parameter settings. The `setup` table contains, for each implementation, which parameter values where used in a specific run. The table `input` contains for each implementation all parameters and their default values. The table `input_setting` contains for every setup the values of the parameters.

The tables `dataset` and `evaluation` both contain data, which can serve as input or output of a `run`. These are linked together by the linking tables `input_data` and `output_data`. Entries in the `dataset` table can be either user-submitted datasets or files containing the result of a run, such as predictions. For each evaluation measure performed, an entry is stored in the `evaluation` table. Querying all experiments of a specific type of task is easiest if the inputs and outputs of that task types are combined in a single table. For this reason, the views `SVCRun` and `SVRRun` have been introduced for Supervised Classification tasks and Supervised Regression tasks, respectively. These are materialized views containing all inputs, outputs and results of such an experiment.

For each implementation and dataset, a number of meta-features [6] are

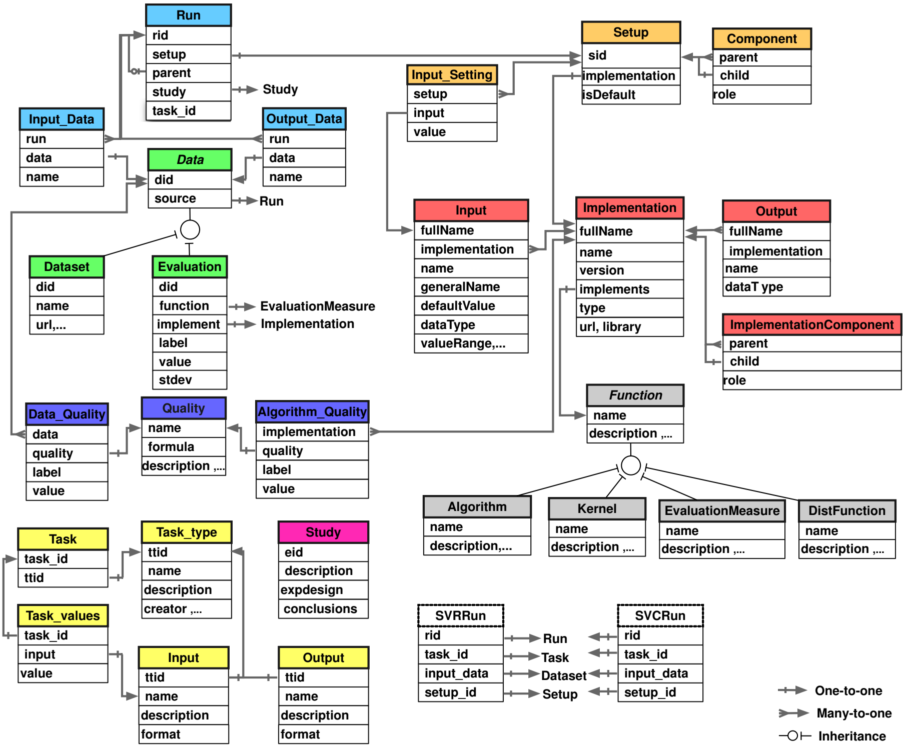


Figure 3: Database schema.

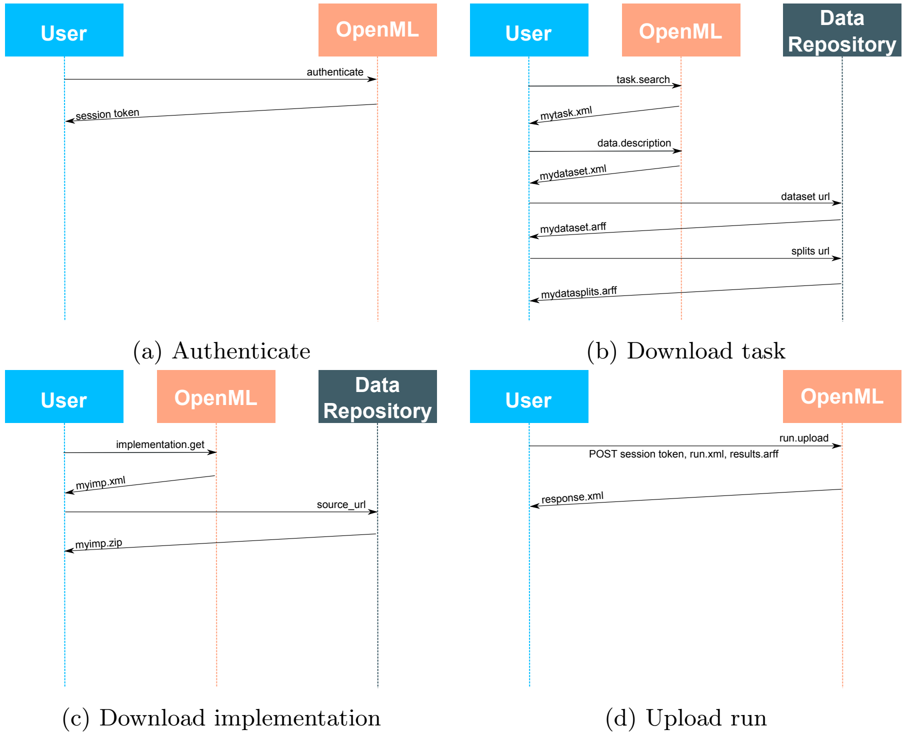
obtained and stored in the `data_quality` and `algorithm_quality` table, respectively. These meta-features are called *qualities*. A list of all qualities can be found in their corresponding tables.

5 RESTful API

In order to enable the sharing of experiments, a web API[5] has been developed. The API contains functions that facilitate downloading datasets, tasks and implementations. Furthermore, it enables the uploading of datasets, implementations and runs. The API also contains functions that list evaluation measures, licence data and evaluation methods. We will briefly explain the most important features.

Functions that involve the uploading of content require the user to provide a *session hash*. A session hash is a unique string which is used to authenticate

[5] Full documentation of the API can be found at `http://www.openml.org/api/`



(a) Authenticate

(b) Download task

(c) Download implementation

(d) Upload run

Figure 4: Use case diagrams of the API.

the user. It is valid for a limited amount of time. Users can obtain a session hash by invoking the function `openml.authenticate` (see also Figure 4a). Inputs for this function are the username and an MD5 hash of the passwords.

Tasks can be obtained by invoking the function `openml.task.search`. An XML file, similar to the XML file shown in Figure 2, is returned. The source data described is an ID referring to a dataset. In order to obtain information concerning this dataset, including a download URL, users should perform an additional call to `openml.data.description`. The dataset can reside in any data repository, including a user's personal webpage. Figure 4b details on how the content of a task is obtained.

Datasets and implementations can be obtained using the API. Both are referred to with an ID. By invoking the functions `openml.data.description` and `openml.implementation.get` with this ID as parameter, users obtain an XML file describing the dataset or implementation. Figure 4c shows how to download an implementation. Datasets can be obtained in a similar way.

Runs can be submitted by invoking the function `openml.run.upload`. Figure 4d outlines how this works. The user provides an XML file describing which implementation was used, and what the parameter settings were. The implementation that was used should already be registered on OpenML. Furthermore, all output of the run must be submitted. For the supervised classification and regression tasks, this will include a file with predictions, which will be evaluated on the server and stored in the database. The server will return an ID referring to the record in the database. Uploading datasets and implementations happens in a similar way. For this the functions `openml.data.upload` and `openml.implementation.upload` are used, respectively.

A list of all evaluation measures for usage in tasks can be obtained by invoking `openml.evaluation.measures`.

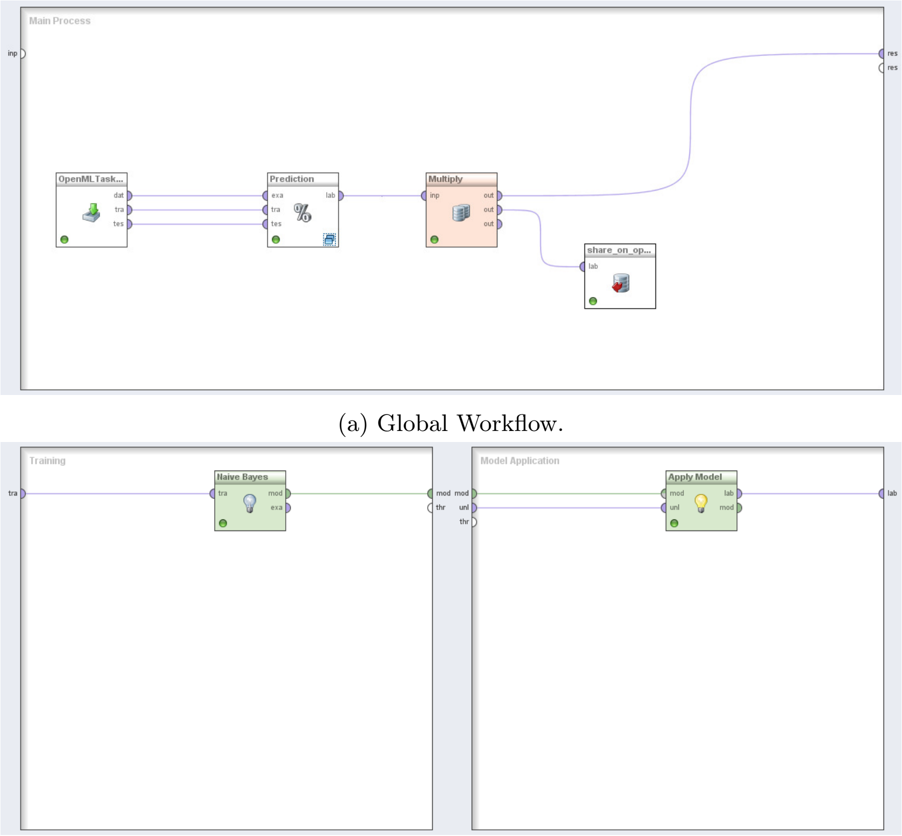
6 Sharing Experiments

To facilitate the sharing of experiments, plugins are being developed for popular data mining and machine learning tools, including RapidMiner. The RapidMiner plugin can be downloaded from the OpenML website. It introduces three new operators.

The *Read OpenML Task* operator handles the downloading of tasks. When presented with a task id, it automatically downloads this task and all associated content, i.e., the input dataset and the data splits. Every entity downloaded from OpenML is cached on the user's local disk. The operator composes the various training and test sets, and marks attributes with certain roles as such, e.g. the target attribute or a row id.

The resulting training and test set will be sent to the *OpenML Prediction* operator. For each training set submitted to this operator, a predictive model is built, which generates predictions for all instances in the test set. These predictions will be sent to the *Share on OpenML* operator, which is responsible for submitting the results to OpenML. First, it checks whether the implementation already exists, and if not, it will be registered. After that, all parameter values are tracked. Finally, an XML file describing the run and an ARFF file containing the predictions will be sent to OpenML.

Figure 5 contains a workflow which uses these operators. This workflow can also be downloaded from the website. Before it can be run, a local directory for caching the downloaded data is required. This can be done in the Preferences menu, under the OpenML tab. When this is set, a workflow containing the OpenML operators can be created. A global outline is shown in Figure 5a. The operators are connected to each other in a straightforward way. We used a *multiply* operator to split the outcome of the Prediction operator to both the general output and the Share on OpenML operator. Note that if the user does not want to share his results on line, he can simply omit the Share on OpenML operator.



(a) Global Workflow.

(b) Sub workflow of the OpenML Prediction operator.

Figure 5: Workflow which downloads an OpenML task, and sends back the results.

By clicking on the OpenML Prediction operator, a screen similar to Figure 5b is shown. This is where subworkflows can be created, to handle both the training and the test data. As for the subworkflow that handles the training data, make sure that at least a model is created, e.g., by including a *Naive Bayes* operator. For the Model application part it typically suffices to insert an *Apply Model* operator. Finally, as parameter of the Read OpenML Task operator, a task id should be provided. These can be searched from the OpenML website.

7 Searching OpenML

All experiments in the database are openly available to everyone. Several ways of searching through these experiments are provided.[6] The most notable ways of searching through OpenML are textual search, the "search runs" interface and the SQL interface.

All implementations, datasets and evaluation metrics submitted to OpenML are required to include meta-data, such as a name, textual description, licence data and in the case of implementations, installation notes and dependencies. These textual descriptions are indexed by a search engine running on the website, so that implementations and datasets can be searched through keywords.

The "search runs" is a wizard interface specialized in benchmark queries. It can be found under the 'Advanced' tab of the search page. The user is presented with a form where he specifies which datasets (or collections of datasets) and implementations he is interested in. Furthermore, he specifies on which evaluation measure the benchmark should be performed. Typical questions that can be answered with this interface are "what implementation performs best on dataset X", "compare several implementations on all datasets", "show the effect of data property DP on the optimal value of parameter P" and "how influence parameter settings the performance of implementation X".

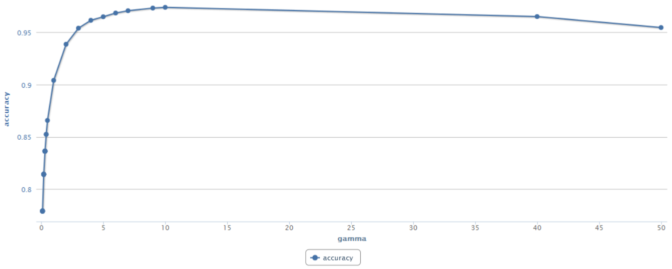
The most flexible way of searching through OpenML is querying the database directly by means of SQL statements. With some knowledge of the database (see Section 4) complex queries can be executed in any way the user wants it. Under the 'Advanced' tab of the search page some queries are provided. The user can also inspect the SQL code of these queries, so these can be adapted to the user's specific needs. In Figure 6 an example of such a query is provided. It studies the effect of the gamma parameter of the Weka implementation of a Support Vector Machine, on the UCI letter dataset [2].

The results of queries can be obtained in CSV and ARFF format. Furthermore, scatterplots and line plots (as shown in Figure 6b) are provided.

[6] `http://www.openml.org/search/`

```
SELECT ps.value as gamma, e.value as accuracy
FROM cvrun r, algorithm_setup s, function_setup kernel, dataset d,
     input_setting ps, evaluation e
WHERE r.learner=s.sid and s.algorithm='SVM' AND kernel.parent=s.sid
    AND kernel.function='RBFKernel' AND ps.setup=s.sid AND ps.input='
     weka.SMO(1.53.2.2)_G' AND e.source=r.rid AND e.function='
     predictive_accuracy' AND r.inputdata=d.did AND d.name='letter'
```

(a) SQL statement



(b) Line plot of the result

Figure 6: Studying the effect of a parameter.

8 Summary

OpenML aims to stimulate an open approach to machine learning research, by collecting results in a database. In order to provide an easy way of sharing these, plugins for various machine learning tools will be provided, including RapidMiner. Instead of running experiments over and over again, users can easily query the database and obtain the results on relevant research questions.

Future work on OpenML includes the integration with other machine learning platforms, such as MLData and myExperiment. Also, the support for a broader range of task types, such as time series analyses, feature selection and graph mining, will be provided. Future work on the RapidMiner plugin includes a better integration with the various services of OpenML. Currently, the plugin is mainly focussed on downloading tasks and uploading results. Features like downloading workflows, uploading datasets and inspecting results could be valuable additions to the plugin.

Acknowledgments

This work is supported by grant 600.065.120.12N150 from the Dutch Fund for Scientific Research (NWO), and by the IST Programme of the European Community, under the Harvest Programme of the PASCAL2 Network of Excellence, IST-2007-216886.

References

[1] D. De Roure, C. Goble, and R. Stevens. The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 25:561–567, 2009.

[2] P.W. Frey and D. J. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161, 1991.

[3] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(suppl 2):W677–W682, 2010.

[4] H. Hirsh. Data mining research: Current status and future opportunities. *Statistical Analysis and Data Mining*, 1(2):104–107, 2008.

[5] M. A. Nielsen. The Future of Science: Building a Better Collective Memory. *APS Physics*, 17(10), 2008.

[6] Y. Peng, P. Flach, C. Soares, and P. Brazdil. Improved Dataset Characterisation for Meta-Learning. *Lecture Notes in Computer Science*, 2534:141–152, 2002.

[7] J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes. Experiment databases. A new way to share, organize and learn from experiments. *Machine Learning*, 87(2):127–158, 2012.