16

Metadata Repositories

Summary. This chapter presents a review of online repositories where researchers can share data, code, and experiments. In particular, it covers OpenML, an online platform for sharing and organizing machine learning data automatically. OpenML contains thousands of datasets and algorithms, and millions of experimental results. We describe the basic philosophy involved, and its basic components: datasets, tasks, flows, setups, runs, and benchmark suites. OpenML has API bindings in various programming languages, making it easy for users to interact with the API in their native language. One important feature of OpenML is the integration into various machine learning toolboxes, such as Scikit-learn, Weka, and mlR. Users of these toolboxes can automatically upload all their results, leading to a large repository of experimental results.

16.1 Introduction

All around the globe, thousands of machine learning experiments are being executed on a daily basis, generating a constant stream of empirical information on machine learning techniques. If we could capture, store, and organize all these results, they would provide a rich and versatile resource for many different metalearning applications.

This chapter covers OpenML (Vanschoren et al., 2014), an online platform for machine learning researchers to share and organize machine learning data automatically and in fine detail. OpenML engenders a dynamic approach to experimentation, in which experiments can be freely shared, linked together, and immediately reused by researchers all over the world. At the same time, it keeps a detailed log of all metadata about the datasets, algorithms, and experimental results, which enables us to learn across all these datasets and experiments, and then transfer this information when learning new tasks.

16.2 Organizing the World Machine Learning Information

Paradoxically, while the machine learning community so greatly values the proper collection of data to allow trustworthy analysis, there is surprisingly little work on systematically collecting and organizing the outputs of machine learning experiments in a way that allows meta-level learning.

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_16

16.2.1 The need for better metadata

This has a profound effect on progress in machine learning. Indeed, without access to reusable prior experiments to build on, each study has to start from scratch. In practice, this limits the depth of many studies, which makes them less generalizable, less interpretable, or even downright contradictory or biased (Aha, 1992; Hand, 2006; Keogh and Kasetty, 2003; Hoste and Daelemans, 2005; Perlich et al., 2003). This makes it very hard for us as a community to correctly interpret the literature and guide future work. Moreover, the competitive mindset of machine learning research often focuses more on small studies dominating the state of the art, rather than large-scale, rigorous and informative analysis (Sculley et al., 2018). The way that we run machine learning experiments is also full of undocumented assumptions and decisions, and a lot of this detail never makes it into papers. As a result, machine learning is grappling with a reproducibility crisis (Hutson, 2018; Hirsh, 2008).

Machine learning experiments can be very sensitive to their exact inputs, such as the exact training sets, hyperparameter settings, implementation details, and evaluation procedures. It is therefore crucial to log and document these exactly. If even humans grapple to understand the exact meaning and truthfulness of empirical results, then metalearning techniques that automatically learn from this data can be very easily misled. If we want to have any chance of obtaining deep and generalizable metalearning results, we first need to collect and organize extensive, finely detailed, and correct metadata. This goes beyond the resources of any individual researcher. It is a community-wide effort that requires good practices as well as the necessary tools to systematically gather detailed empirical data and pushing it to online platforms that help us structure and reuse the world's machine learning information.

16.2.2 Tools and initiatives

Several initiatives do aim to partially alleviate these problems, for instance through the creation of dataset repositories. The UCI repository (Dheeru and Taniskidou, 2017) and LIBSVM (Chang and Lin, 2011) offer a wide range of datasets. Many more focused repositories also exist, such as UCR (Chen et al., 2015) for time series data and Mulan (Tsoumakas et al., 2011) for multilabel datasets. More recent initiatives additionally provide programmatic access to datasets, such as the `kaggle.com` and PMLB (Olson et al., 2017) APIs for downloading general (mostly classification) datasets, the KEEL (Alcala et al., 2010) API for imbalanced classification and datasets with missing values, and the skdata (Bergstra et al., 2015) API for downloading computer vision and natural language processing datasets.

Other platforms additionally link datasets to reproducible experiments.[1] Early initiatives include StatLog (Michie et al., 1994), MetaL (Brazdil et al., 2009), DELVE,[2] and MLcomp,[3] but none of these projects is still being maintained. Experiment databases for machine learning (Vanschoren et al., 2012), the forerunners of OpenML, were among the first to organize large amounts of empirical results and make them queryable through online interfaces, but they required users to manually translate their experiments into a common format, which was tedious and prone to errors and omissions. More recently,

[1]Data mining challenge platforms, such as `kaggle.com`, `chalearn.org`, and `aicrowd.com` do share leaderboard results, but these are often not reproducible.

[2]`http://www.cs.toronto.edu/~delve/`

[3]`mlcomp.org`

the OpenAI Gym (Brockman et al., 2016) allows running and evaluating reproducible reinforcement learning experiments, but lacks rich, organized metadata about complete training episodes. AI benchmarking systems such as MLPerf[4] and reproducibility initiatives such as PapersWithCode[5] do provide very interesting evaluation results but are often too targeted to provide general metadata for subsequent metalearning. It is important to note, though, that reproducibility is a key requirement for metalearning, since we need to be able to trust the metadata on which we build.

16.3 OpenML

Vanschoren et al. (2014) introduced the OpenML project, an online platform for sharing datasets and reproducible experiments. OpenML provides APIs (in Python, Java, and R) for downloading data in uniform formats into popular machine learning libraries, and uploading and comparing the ensuing results. It also provides metadata for standardizing evaluations (e.g., predefined train-test splits) and for in-depth analysis of evaluation results.

OpenML is integrated into various popular machine learning tools, such as Weka (Hall et al., 2009), R (Bischl et al., 2016b), Scikit-learn (Buitinck et al., 2013; Pedregosa et al., 2011), and MOA (Bifet et al., 2010a), and several more integrations are in progress, including deep learning tools. This allows anyone to easily import datasets into these tools, pick any algorithm or workflow to run, and automatically share all obtained results. Results are being produced locally: everyone that participates can run experiments locally (or anywhere they want) and afterwards share the results on OpenML. The web interface provides easy access to all collected data and code, compares all results obtained on the same data or algorithms, builds data visualizations, and supports online discussions.

OpenML offers various services to share and find datasets, to download or create scientific *tasks*, to share and find algorithm pipelines (called *flows*), and to share and organize experiments (called *runs*).

16.3.1 Datasets

For many metalearning applications, every dataset is a single metadata point. Hence, it is crucial to provide a rich and growing set of varied datasets. OpenML allows datasets to be uploaded directly from the environments where they were created (e.g., from a Python script). The data can be uploaded by a single API call, or simply referenced by a URL. This URL may be a landing page with further information or terms of use, or it may be an API call to large repositories of scientific data such as the SDSS (Szalay et al., 2002). OpenML will automatically version each dataset and make sure that empirical results are linked to specific versions. Authors can license their data and add citation requests. Finally, extra information can be added, such as the (default) target attribute(s) in labeled data, or the row-id attribute for data where instances are named.

Next, OpenML will compute an array of data characteristics, also called *metafeatures*, often categorized as either simple, statistical, information theoretic or landmarkers (Pfahringer et al., 2000). Table 16.1 shows some of the metafeatures computed by OpenML.

[4]https://mlperf.org/
[5]https://paperswithcode.com/

Table 16.1: Standard metafeatures that are available in OpenML. Table taken from van Rijn (2016).

Category	Metafeatures
Simple	# Instances, # Attributes, # Classes, Dimensionality, Default Accuracy, # Observations with Missing Values, # Missing Values, % Observations with Missing Values, % Missing Values, # Numeric Attributes, # Nominal Attributes, # Binary Attributes, % Numeric Attributes, % Nominal Attributes, % Binary Attributes, Majority Class Size, % Majority Class, Minority Class Size, % Minority Class
Statistical	Mean of Means of Numeric Attributes, Mean Standard Deviation of Numeric Attributes, Mean Kurtosis of Numeric Attributes, Mean Skewness of Numeric Attributes
Information theoretic	Class Entropy, Mean Attribute Entropy, Mean Mutual Information, Equivalent Number of Attributes, Noise to Signal Ratio
Landmarkers	Accuracy of Decision Stump, Kappa of Decision Stump, Area under the ROC Curve of Decision Stump, Accuracy of Naive Bayes, Kappa of Naive Bayes, Area under the ROC Curve of Naive Bayes, Accuracy of k-NN, Kappa of k-NN, Area under the ROC Curve of k-NN, ...

The datasets and their metadata can be retrieved via an online search engine, via the REST API as JSON and XML, and as native Python, R, or Java data structures using the corresponding APIs. This structured metadata includes user-provided descriptions, extracted attribution information, metafeatures, and even statistics of the data distribution.

16.3.2 Task types

A dataset alone does not constitute a scientific task. We must first agree on what types of results are expected to be shared. This is expressed in *task types*: they define what types of inputs are given, which types of output are expected to be returned, and what scientific protocols should be used. For instance, classification tasks should include well-defined cross-validation procedures and labeled input data and require predictions as outputs.

OpenML currently covers supervised classification, supervised regression, clustering, learning curve analysis, data stream classification, survival analysis, and subgroup discovery. These are very generally defined: a classification can cover text, image, or any other type of classification. Each task types comes with information on how models should be trained and evaluated, such as predefined cross-validation splits for classification, and prequential (test-then-train) splits for data streams.

16.3.3 Tasks

Tasks are instantiations of task types with specific inputs. An example of such a task is shown in Figure 16.1. In this case, it is a classification task defined on the MNIST dataset (version 1). Next to the dataset, the task includes the target attribute and the evaluation

Given inputs

source_data	anneal (1)	Dataset (required)
estimation_procedure	10-fold Cross-validation	EstimationProcedure (required)
evaluation_measures	predictive_accuracy	String (optional)
target_feature	class	String (required)
data_splits	http://www.openml.org/api_splits/get/1/1/Task_1_splits.arff	TrainTestSplits (hidden)

Expected outputs

model	A file containing the model built on all the input data	File (optional)
evaluations	A list of user-defined evaluations of the task as key-value pairs	KeyValue (optional)
predictions	An arff file with the predictions of a model	Predictions (required)

Fig. 16.1: Example of an OpenML task description

procedure (here, 10-fold cross-validation) used to generate the train and test splits. The required outputs for this task are the predictions for all test instances and, optionally, the models built and evaluations calculated by the user. OpenML will always compute a large range of evaluation measures on the server to ensure objective comparison. The preferred evaluation measure can be selected afterwards depending on the type of meta-level analysis.

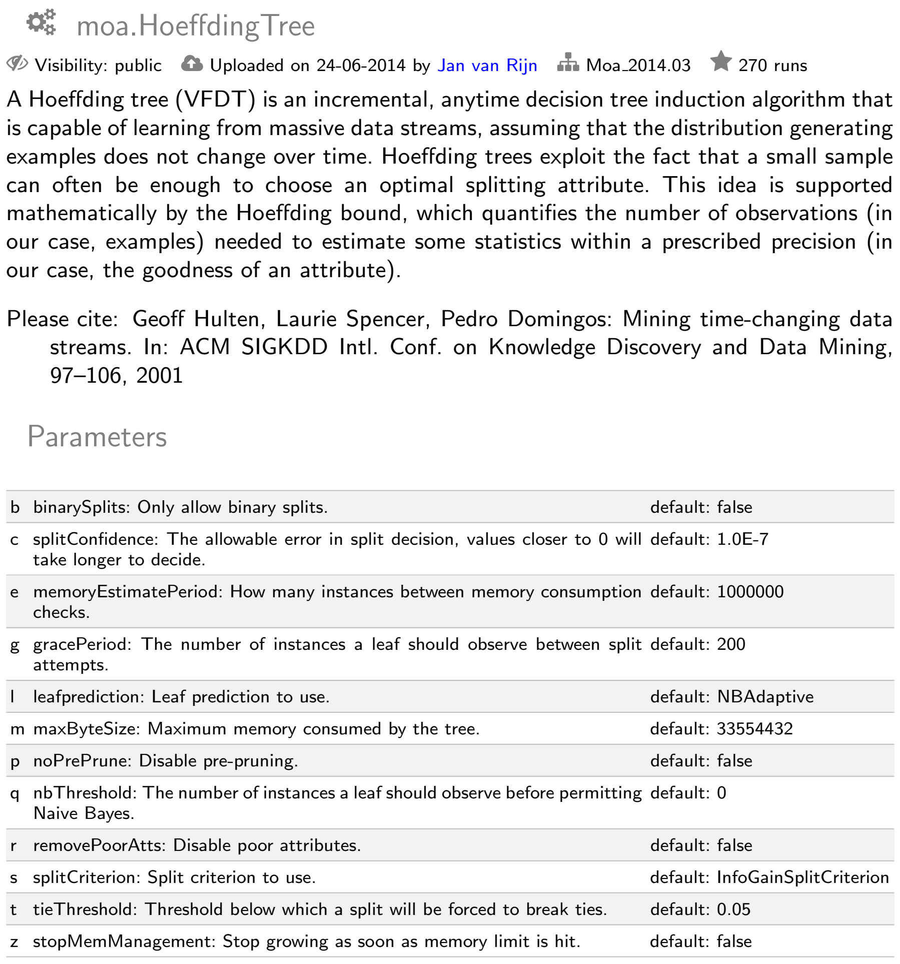
Tasks can again be viewed or downloaded through the website, REST API, or language-specific APIs, including a list of all algorithms trained on that task and their evaluations. These can then be reused in different metalearning applications.

16.3.4 Flows

Flows are implementations of single algorithms, workflows (also known as pipelines), or scripts designed to solve a given task. For actual pipelines, the flows will define the exact components of the pipelines and how they fit together, as well as details about each component, such as the list of hyperparameters that could be tuned. For each hyperparameter, the name, description, and default value (if known) are stored. OpenML also stores metadata such as the dependencies and citation information, to enable the flows to be rebuilt and reused later on.

Flows can be updated as often as needed. OpenML will version each uploaded flow, while users can provide their own version name for reference. As with datasets, each flow has its own page which combines all known information and all results obtained by running the flow on OpenML tasks (see Figure 16.2).

It is important to emphasize that flows are typically not executed on the OpenML server. They can be executed locally or on any computer server the user has access to.



moa.HoeffdingTree

Visibility: public Uploaded on 24-06-2014 by Jan van Rijn Moa_2014.03 270 runs

A Hoeffding tree (VFDT) is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the goodness of an attribute).

Please cite: Geoff Hulten, Laurie Spencer, Pedro Domingos: Mining time-changing data streams. In: ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 97–106, 2001

Parameters

b	binarySplits: Only allow binary splits.	default: false
c	splitConfidence: The allowable error in split decision, values closer to 0 will take longer to decide.	default: 1.0E-7
e	memoryEstimatePeriod: How many instances between memory consumption checks.	default: 1000000
g	gracePeriod: The number of instances a leaf should observe between split attempts.	default: 200
l	leafprediction: Leaf prediction to use.	default: NBAdaptive
m	maxByteSize: Maximum memory consumed by the tree.	default: 33554432
p	noPrePrune: Disable pre-pruning.	default: false
q	nbThreshold: The number of instances a leaf should observe before permitting Naive Bayes.	default: 0
r	removePoorAtts: Disable poor attributes.	default: false
s	splitCriterion: Split criterion to use.	default: InfoGainSplitCriterion
t	tieThreshold: Threshold below which a split will be forced to break ties.	default: 0.05
z	stopMemManagement: Stop growing as soon as memory limit is hit.	default: false

Fig. 16.2: Example of an OpenML flow

16.3.5 Setups

A setup is the combination of a flow and a certain configuration of the hyperparameters. Setups allow for analysis on the effect of hyperparameters, as is shown in Chapter 17. Setups that are run with default parameter settings are flagged as such.

16.3.6 Runs

Runs are applications of flows on a specific task. They are submitted by uploading the required outputs (e.g., predictions) together with the task id, the flow id, and any parameter settings. Each run also has its own page with all details and results, shown partially in Figure 16.3. In this case, it is a classification run, where the predictions of the specific task are uploaded, and the evaluation measures are calculated on the server. Based on the parameter settings, the run is also linked to a setup.

OpenML calculates detailed evaluation results, including per-fold predictions. For class-specific measures such as area under the ROC curve, precision, and recall, per-class results are stored. Additional information, such as run times and details on hardware, can be provided by the user.

Because each run is linked to a specific task, flow, setup, and author, OpenML can filter, aggregate, and visualize results accordingly. Depending on the metalearning application, different meta-datasets can be built, including the dataset metafeatures, flow hyperparameter settings, and run evaluation results.

16.3.7 Studies and benchmark suites

Finally, OpenML allows sets of tasks and runs to be bundled. This makes it easier to define specific sets of tasks for a specific goal, and to group a specific set of runs (and their evaluation results) to obtain a fixed set of metadata for subsequent analysis. A set of tasks also implies a specific set of underlying datasets, and a set of runs implies a specific set of flows and tasks.

A special use case for a set of tasks is a *benchmark suite* (Bischl et al., 2021), a set of tasks that are selected to evaluate algorithms under a precisely specified set of conditions. They allow experiments run on these datasets to be clearly interpretable, comparable, and reproducible. Benchmarking suites can be created and retrieved using the existing OpenML interfaces and APIs. A first example of such a suite is the OpenML-CC18 (Bischl et al., 2021). Subsequently, these tasks could be easily downloaded and then classifiers could be run on them using different libraries, including Scikit-learn (Pedregosa et al., 2011), mlr (Bischl et al., 2016a), and Weka (Hall et al., 2009), through existing OpenML bindings (Casalicchio et al., 2017; van Rijn et al., 2015; Feurer et al., 2019a).

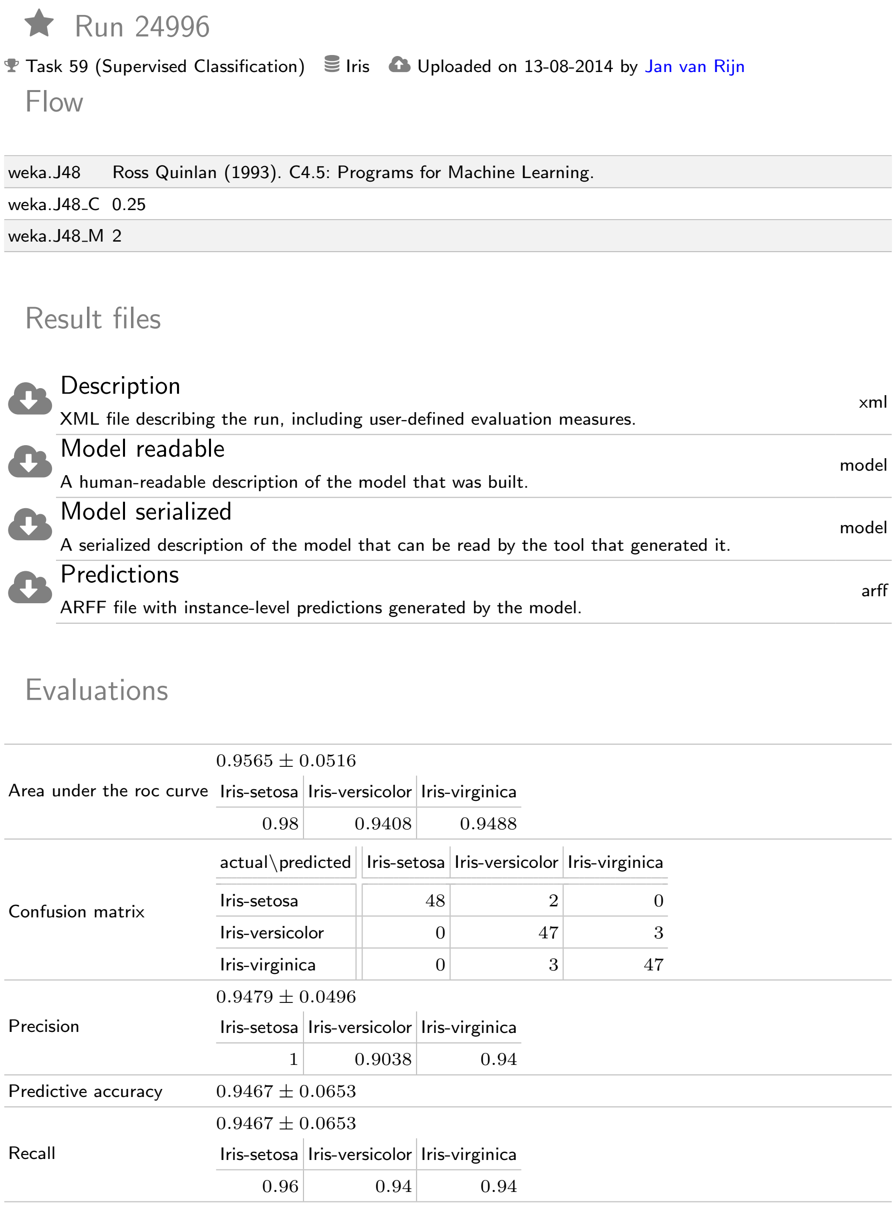
16.3.8 Integrations of OpenML in machine learning environments

OpenML is integrated into several popular machine learning environments, so that it can be used out of the box. These integrations are offered either as libraries (e.g., R or Python packages) or as plugins for existing toolboxes.

Figure 16.4 shows how OpenML is integrated in WEKA's Experimenter (Hall et al., 2009). After selecting OpenML as the result destination and providing login credentials, a number of tasks can be added through a dialogue. The plug-in supports the use of filters (for preprocessing operations), uploading of parameter sweep traces (for parameter optimization), and uploading of human-readable model representations produced by WEKA.

Other integrations include MOA (Bifet et al., 2010a), for running experiments and doing metalearning on data streams (Bifet et al., 2010b; Read et al., 2012), and RapidMiner (Hofmann and Klinkenberg, 2013), for running complex workflows (van Rijn and Vanschoren, 2015).

Researchers who use R or Python can use the *openml* package, provided on central repository databases CRAN and PyPI. Figure 16.5 is an example showing how to download a task and upload a run in R (together with all metadata). Once a classifier is created (line 3), it takes two function calls to download the task into memory (line 4) and run



Run 24996

Task 59 (Supervised Classification) Iris Uploaded on 13-08-2014 by Jan van Rijn

Flow

weka.J48	Ross Quinlan (1993). C4.5: Programs for Machine Learning.
weka.J48_C	0.25
weka.J48_M	2

Result files

Description XML file describing the run, including user-defined evaluation measures.	xml
Model readable A human-readable description of the model that was built.	model
Model serialized A serialized description of the model that can be read by the tool that generated it.	model
Predictions ARFF file with instance-level predictions generated by the model.	arff

Evaluations

Area under the roc curve: 0.9565 ± 0.0516

Iris-setosa	Iris-versicolor	Iris-virginica
0.98	0.9408	0.9488

Confusion matrix:

actual\predicted	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	48	2	0
Iris-versicolor	0	47	3
Iris-virginica	0	3	47

Precision: 0.9479 ± 0.0496

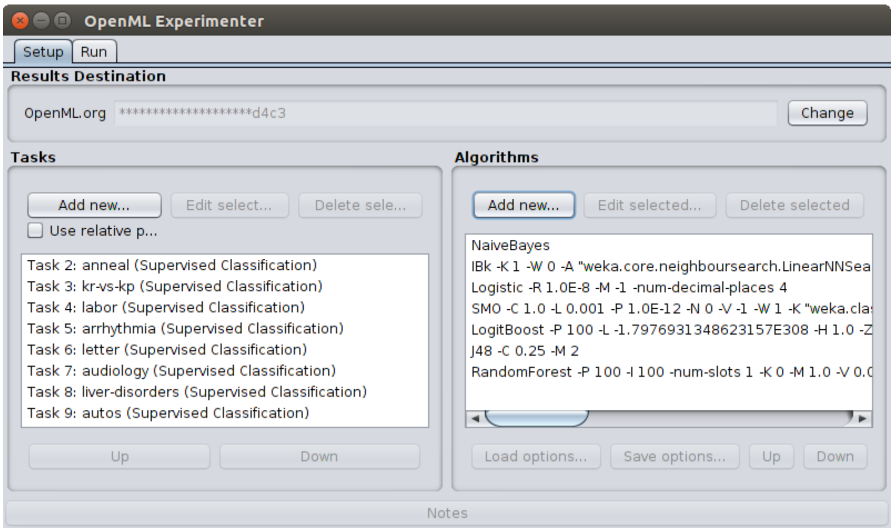
Iris-setosa	Iris-versicolor	Iris-virginica
1	0.9038	0.94

Predictive accuracy: 0.9467 ± 0.0653

Recall: 0.9467 ± 0.0653

Iris-setosa	Iris-versicolor	Iris-virginica
0.96	0.94	0.94

Fig. 16.3: Example of an OpenML run


OpenML Experimenter

Setup | Run

Results Destination

OpenML.org ******************d4c3 Change

Tasks

Add new... | Edit select... | Delete sele...

Use relative p...

```
Task 2: anneal (Supervised Classification)
Task 3: kr-vs-kp (Supervised Classification)
Task 4: labor (Supervised Classification)
Task 5: arrhythmia (Supervised Classification)
Task 6: letter (Supervised Classification)
Task 7: audiology (Supervised Classification)
Task 8: liver-disorders (Supervised Classification)
Task 9: autos (Supervised Classification)
```

Up | Down

Algorithms

Add new... | Edit selected... | Delete selected

```
NaiveBayes
IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSea
Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.cla
LogitBoost -P 100 -L -1.7976931348623157E308 -H 1.0 -Z
J48 -C 0.25 -M 2
RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.0
```

Load options... | Save options... | Up | Down

Notes

Fig. 16.4: WEKA integration of OpenML

the classifier (line 5). After the classifier has been applied, it takes a single function call to upload all the results (line 6).

```
1 library(mlr)
2 library(OpenML)
3 lrn = makeLearner("classif.randomForest")
4 task = getOMLTask(6)
5 run = runTaskMlr(task, lrn)
6 uploadOMLRun(run)
```

Fig. 16.5: R code to run a random forest classifier as implemented in mlR on the letter task (task id = 6)

The Python code is very similar (see Figure 16.6) and works very similarly to the R code. The function calls `get_task()` and `run_model_on_task` are used to download the task into memory (line 4) and run the classifier on the task (line 5). Finally, the member function `run.publish` uploads the results to OpenML (line 6).

As usual, the Java code is a bit more verbose (see Figure 16.7). For brevity, the imports in the header have been omitted. The Java function `executeTask` runs the classifier on the task and automatically uploads each executed run to the server (line 8).

These APIs also allow for convenient downloading of all results in various formats.

```python
from sklearn import ensemble
from openml import tasks, runs
clf = ensemble.RandomForestClassifier()
task = tasks.get_task(6)
run = runs.run_model_on_task(clf, task)
run.publish()
```

Fig. 16.6: Python code to run a random forest classifier as implemented in Scikit-learn on the letter task (task id = 6)

```java
public static void runTasksAndUpload() throws Exception {
  OpenmlConnector openml = new OpenmlConnector();
  openml.setApiKey("FILL_IN_OPENML_API_KEY");
  Classifier forest = new RandomForest();
  Task task = openml.taskGet(6);
  Instances d = InstancesHelper.getDatasetFromTask(openml, task
    );
  Pair<Integer, Run> result = RunOpenmlJob.executeTask(
    openml, new WekaConfig(), task.getTask_id(), forest);
  Run run = openml.runGet(result.getLeft());
}
```

Fig. 16.7: Java code to run a random forest classifier as implemented in Weka on the letter task (task id = 6)

Example of one study exploiting existing evaluation results

Figure 16.8 is an example of Python code showing how detailed evaluation results can be downloaded to study the effect of hyperparameters of the SVM algorithm. The resulting plots are shown in Figure 16.9.

In the next chapter, we explore how the experimental data available in OpenML can be used to gain new insights into the relationship between properties of data, workflows, and performance.

References

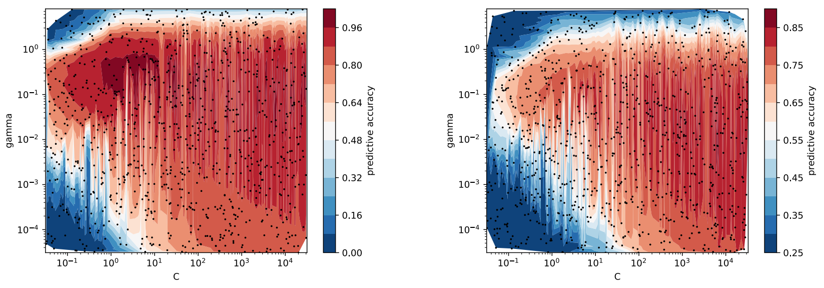
Aha, D. W. (1992). Generalizing from case studies: A case study. In Sleeman, D. and Edwards, P., editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML92)*, pages 1–10. Morgan Kaufmann.

Alcala, J., Fernandez, A., Luengo, J., Derrac, J., Garcia, S., Sanchez, L., and Herrera, F. (2010). Keel datamining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287.

Bergstra, J., Pinto, N., and Cox, D. (2015). SkData: data sets and algorithm evaluation protocols in Python. *Computational Science & Discovery*, 8(1).

```python
import openml;
import numpy as np
import matplotlib.pyplot as plt
df = openml.evaluations.list_evaluations_setups(
    'predictive_accuracy', flow=[8353], task=[6],
    output_format='dataframe',
    parameters_in_separate_columns=True,
) # Choose SVM flow (e.g. 8353) and dataset 'letter' (task 6).
hp_names = ['sklearn.svm.classes.SVC(16)_C','sklearn.svm.
    classes.SVC(16)_gamma']
df[hp_names] = df[hp_names].astype(float).apply(np.log)
C, gamma, score= df[hp_names[0]], df[hp_names[1]], df['value']
cntr = plt.tricontourf(
  C, gamma, score, levels=12, cmap='RdBu_r')
plt.colorbar(cntr, label='accuracy')
plt.xlim((min(C), max(C))); plt.ylim((min(gamma), max(gamma)))
plt.xlabel('C (log10)', size=16);
plt.ylabel('gamma (log10)', size=16)
plt.title('SVM performance landscape', size=20)
```

Fig. 16.8: Code for retrieving the predictive accuracy of a SVM classifier on the “letter” dataset and creating a contour plot (retrieved from Feurer et al. (2019b))



(a) “Letter” dataset (b) “Vehicle” dataset

Fig. 16.9: Surface plots of the *gamma* and *C* hyperparameter of SVM classifier. Both axes show the value of a hyperparameter, and the color of the grid shows how well a certain configuration performed (retrieved from Feurer et al. (2019b))

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010a). MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11:1601–1604.

Bifet, A., Holmes, G., and Pfahringer, B. (2010b). Leveraging Bagging for Evolving Data Streams. In *Machine Learning and Knowledge Discovery in Databases*, volume 6321 of *Lecture Notes in Computer Science*, pages 135–150. Springer.

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2021). OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, NIPS'21.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., et al. (2016a). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016b). mlr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5.

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. (2009). *Metalearning: Applications to data mining*. Springer.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv:1606.01540*.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., and Bischl, B. (2017). OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*.

Chang, C. C. and Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., and Batista, G. (2015). The UCR time series classification archive. `www.cs.ucr.edu/~eamonn/time_series_data/`.

Dheeru, D. and Taniskidou, E. K. (2017). UCI machine learning repository.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2019a). Auto-sklearn: Efficient and robust automated machine learning. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 113–134. Springer.

Feurer, M., van Rijn, J. N., Kadra, A., Gijsbers, P., Mallik, N., Ravi, S., Müller, A., Vanschoren, J., and Hutter, F. (2019b). OpenML-Python: an extensible Python API for OpenML. *arXiv preprint arXiv:1911.02490*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

Hand, D. (2006). Classifier technology and the illusion of progress. *Statistical Science*, 21(1):1–14.

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

Hofmann, M. and Klinkenberg, R. (2013). *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Data Mining and Knowledge Discovery. Chapman & Hall/CRC.

Hoste, V. and Daelemans, W. (2005). Comparing learning approaches to coreference resolution. There is more to it than bias. In Giraud-Carrier, C., Vilalta, R., and Brazdil, P., editors, *Proceedings of the ICML 2005 Workshop on Meta-Learning*, pages 20–27.

Hutson, M. (2018). Missing data hinder replication of artificial intelligence studies. *Science*.

Keogh, E. and Kasetty, S. (2003). On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. (2017). PMLB: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(36).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., and Dubourg, V. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Perlich, C., Provost, F., and Simonoff, J. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4:211–255.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning*, ICML'00, pages 743–750.

Read, J., Bifet, A., Pfahringer, B., and Holmes, G. (2012). Batch-Incremental versus Instance-Incremental Learning in Dynamic and Evolving Data. In *Advances in Intelligent Data Analysis XI*, pages 313–323. Springer.

Sculley, D., Snoek, J., Wiltschko, A., and Rahimi, A. (2018). Winner's curse? on pace, progress, and empirical rigor. In *Workshop of the International Conference on Representation Learning (ICLR)*.

Szalay, A. S., Gray, J., Thakar, A. R., Kunszt, P. Z., Malik, T., Raddick, J., Stoughton, C., and vandenBerg, J. (2002). The SDSS SkyServer: public access to the Sloan digital sky server data. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 570–581. ACM.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., and Vlahavas, I. (2011). MULAN: A Java library for multi-label learning. *JMLR*, pages 2411–2414.

van Rijn, J. N. (2016). *Massively collaborative machine learning*. PhD thesis, Leiden University.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2015). Having a Blast: Meta-Learning and Heterogeneous Ensembles for Data Streams. In *2015 IEEE International Conference on Data Mining (ICDM)*, pages 1003–1008. IEEE.

van Rijn, J. N. and Vanschoren, J. (2015). Sharing RapidMiner workflows and experiments with OpenML. In Vanschoren, J., Brazdil, P., Giraud-Carrier, C., and Kotthoff, L., editors, *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection (MetaSel)*, number 1455 in CEUR Workshop Proceedings, pages 93–103.

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

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.