



Universiteit
Leiden
The Netherlands

Massively collaborative machine learning

Rijn, J.N. van

Citation

Rijn, J. N. van. (2016, December 19). *Massively collaborative machine learning*. IPA Dissertation Series. Retrieved from <https://hdl.handle.net/1887/44814>

Version: Not Applicable (or Unknown)

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/44814>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/44814> holds various files of this Leiden University dissertation

Author: Rijn, Jan van

Title: Massively collaborative machine learning

Issue Date: 2016-12-19

Experiment Databases

Many fields of science have made significant breakthroughs by adopting online tools that help organize, structure and mine information that is too detailed to be printed in journals. In this chapter, we introduce OpenML, an online platform for machine learning researchers to share and organize data in fine detail, so that they can work more effectively, be more visible, and collaborate with others to tackle harder problems. We discuss some important concepts upon which it is built and showcase various types of Machine Learning studies that can be conducted using information from previous experiments.

4.1 Introduction

When Galileo Galilei discovered the rings of Saturn, he did not write a scientific paper. Instead, he wrote his discovery down, jumbled the letters into an anagram, and sent it to his fellow astronomers. This was common practice among respected scientists of the age, including Leonardo, Huygens and Hooke.

The reason was not technological. The printing press was well in use those days and the first scientific journals already existed. Rather, there was little personal gain in letting your rivals know what you were doing. The anagrams ensured that the original discoverer alone could build on his ideas, at least until someone else made the same discovery and the solution to the anagram had to be published in order to claim priority.

This behaviour changed gradually in the late 17th century. Members of the Royal Society realized that this secrecy was holding them back, and that if they all agreed to publish their findings openly, they would all do better [79]. Under the motto “take

nobody's word for it", they established that scientists could only claim a discovery if they published it first, if they detailed their experimental methods so that results could be verified, and if they explicitly gave credit to all prior work they built upon.

Moreover, wealthy patrons and governments increasingly funded science as a profession, and required that findings be published in journals, thus maximally benefiting the public, as well as the public image of the patrons. This effectively created an economy based on *reputation* [79, 96]. By publishing their findings, scientists were seen as trustworthy by their peers and patrons, which in turn led to better collaboration, research funding, and scientific jobs. This new culture continues to this day and has created a body of shared knowledge that is the basis for much of human progress.

Today, however, the ubiquity of the Internet is allowing new, more scalable forms of scientific collaboration. We can now share detailed observations and methods (data and code) far beyond what can be printed in journals, and interact in real time with many people at once, all over the world.

As a result, many sciences have turned to online tools to share, structure and analyse scientific data on a global scale. Such *networked science* is dramatically speeding up discovery because scientists are now able to build directly on each other's observations and techniques, reuse them in unforeseen ways, mine all collected data for patterns, and scale up collaborations to tackle much harder problems. Whereas the journal system still serves as our collective long-term memory, the Internet increasingly serves as our collective *short-term working memory* [97], collecting data and code far too extensive and detailed to be comprehended by a single person, but instead (re)used by many to drive much of modern science.

Many challenges remain, however. In the spirit of the journal system, these online tools must also ensure that shared data is trustworthy so that others can build on it, and that it is in individual scientists' best interest to share their data and ideas. In this chapter, we introduce OpenML, a collaboration platform through which scientists can automatically share, organize and discuss machine learning experiments, data, and algorithms.

4.2 Networked science

Networked science tools are changing the way we make discoveries in several ways. They allow hundreds of scientists to discuss complex ideas online, they structure information from many scientists into a coherent whole, and allow anyone to reuse all collected data in new and unexpected ways.

4.2.1 Designing networked science

Nielsen [97] reviews many examples of networked science, and explains their successes by the fact that, through the interaction of many minds, there is a good chance that someone has just the right expertise to contribute at just the right time:

Designed serendipity Because many scientists have complementary expertise, any shared idea, question, observation, or tool may be noticed by someone who has just the right (micro)expertise to spark new ideas, answer questions, reinterpret observations, or reuse data and tools in unexpected new ways. By scaling up collaborations, such ‘happy accidents’ become ever more likely and frequent.

Dynamic division of labour Because each scientist is especially adept at certain research tasks, such as generating ideas, collecting data, mining data, or interpreting results, any seemingly hard task may be routine for someone with just the right skills, or the necessary time or resources to do so. This dramatically speeds up progress.

Designed serendipity and a dynamic division of labour occur naturally when ideas, questions, data, or tools are broadcast to a large group of people in a way that allows everyone in the collaboration to discover what interests them, and react to it easily and creatively. As such, for online collaborations to scale, online tools must make it practical for anybody to join and contribute any amount at any time. This can be expressed in the following ‘design patterns’ [97]:

- Encourage small contributions, allowing scientists to contribute in (quasi) real time. This allows many scientists to contribute, increasing the cognitive diversity and range of available expertise.
- Split up complex tasks into many small subtasks that can be attacked (nearly) independently. This allows many scientists to contribute individually and according to their expertise.
- Construct a rich and structured information commons, so that people can efficiently build on prior knowledge. It should be easy to find previous work, and easy to contribute new work to the existing body of knowledge.
- Human attention does not scale infinitely. Scientists only have a limited amount of attention to devote to the collaboration, and should thus be able to focus on their interests and filter out irrelevant contributions.
- Establish accepted methods for participants to interact and resolve disputes. This can be an ‘honour code’ that encourages respectable and respectful beha-

viour, deters academic dishonesty, and protects the contributions of individual scientists.

Still, even if scientists have the right expertise or skill to contribute at the right time, they typically also need the right incentive to do so.

This problem was actually solved already centuries ago by establishing a reputation system implemented using the best medium for sharing information of the day, the journal. Today, the internet and networked science tools provide a much more powerful medium, but they also need to make sure that sharing data, code and ideas online is in scientists' best interest.

The key to do this seems to lie in extending the reputation system [97]. Online tools should allow everyone to see exactly who contributed what, and link valuable contributions to increased esteem amongst the users of the tools and the scientific community at large. The traditional approach to do this is to link useful online contributions to authorship in ensuing papers, or to link the reuse of shared data to citation of associated papers or DOI's. Scientific communities are typically small worlds, thus scientists are encouraged to respect such agreements.

Moreover, beyond bibliographic measures, online tools can define new measures to demonstrate the scientific (and societal) impact of contributions. These are sometimes called *altmetrics* or article-level metrics [112]. An interesting example is ArXiv, an online archive of preprints (unpublished manuscripts) with its own reference tracking system (SPIRES). In physics, preprints that are referenced many times have a high status among physicists. They are added to resumes and used to evaluate candidates for scientific jobs. This illustrates that what gets measured, gets rewarded, and what gets rewarded, gets done [97, 100]. If scholarly tools define useful new measures and track them accurately, scientists will use them to assess their peers.

4.3 Machine learning

Machine learning is a field where a more networked approach would be particularly valuable. Machine learning studies typically involve large datasets, complex code, large-scale evaluations and complex models, none of which can be adequately represented in papers. Still, most work is only published in papers, in highly summarized forms such as tables, graphs and pseudo-code. Oddly enough, while machine learning has proven so crucial in analysing large amounts of observations collected by other scientists, the outputs of machine learning research are typically not collected and organized in any way that allows others to reuse, reinterpret, or mine these results to learn new things, e.g., which techniques are most useful in a given application.

4.3.1 Reusability and reproducibility

This makes us duplicate a lot of effort, and ultimately slows down the whole field of machine learning [63, 153]. Indeed, without prior experiments to build on, each study has to start from scratch and has to rerun many experiments. This limits the depth of studies and the interpretability and generalizability of their results [4, 63]. It has been shown that studies regularly contradict each other because they are biased toward different datasets [80], or because they do not take into account the effects of dataset size, parameter optimization and feature selection [71, 105]. This makes it very hard, especially for other researchers, to correctly interpret the results. Moreover, it is often not even possible to rerun experiments because code and data are missing, or because space restrictions imposed on publications make it practically infeasible to publish many details of the experiment setup. This lack of reproducibility has been warned against repeatedly [80, 102, 142], and has been highlighted as one of the most important challenges in data mining research [67].

4.3.2 Prior work

Many machine learning researchers are well aware of these issues, and have worked to alleviate them. To improve reproducibility, there exist repositories to publicly share benchmarking datasets, such as UCI [90]. Moreover, software can be shared on the MLOSS website. There also exists an open source software track in the Journal for Machine Learning Research (JMLR) where short descriptions of useful machine learning software can be submitted. Also, some major conferences have started checking submissions for reproducibility [91], or issue open science awards for submissions that are reproducible (e.g., ECML/PKDD 2013).

Moreover, there also exist experiment repositories. First, meta-learning projects such as StatLog [93] and MetaL [22], and benchmarking services such as MLcomp run many algorithms on many datasets on their servers. This makes benchmarks comparable, and even allows the building of meta-models, but it does require the code to be rewritten to run on their servers. Moreover, the results are not organized to be easily queried and reused.

Second, data mining challenge platforms such as Kaggle [29] and TunedIT [162] collect results obtained by different competitors. While they do scale and offer monetary incentives, they are adversarial rather than collaborative. For instance, code is typically not shared during a competition.

Finally, the *experiment database for machine learning* [153] was introduced, which organizes results from different users and makes them queryable through an online interface. Unfortunately, it does not allow collaborations to scale easily. It requires researchers to transcribe their experiments into XML, and only covers classification

experiments.

While all these tools are very valuable in their own right, they do not provide many of the requirements for scalable collaboration discussed above. It can be quite hard for scientists to contribute, there is often no online discussion, and they are heavily focused on benchmarking, not on sharing other results such as models. By introducing OpenML, we aim to provide a collaborative science platform for machine learning.

4.4 OpenML

OpenML is an online platform where machine learning researchers can automatically share data in fine detail and organize it to work more effectively and collaborate on a global scale [124, 155, 154].

It allows anyone to challenge the community with new data to analyse, and everyone able to mine that data to share their code and results (e.g., models, predictions, and evaluations). In this sense, OpenML is similar to data mining challenge platforms, except that it allows users to work collaboratively, building on each other's work. OpenML makes sure that each (sub)task is clearly defined, and that all shared results are stored and organized online for easy access, reuse and discussion.

Moreover, it is being integrated in popular data mining platforms such as Weka [61], R [16, 148], Scikit-learn [28, 103], RapidMiner [151, 121], MOA [13] and Cortana [92]. This means that anyone can easily import the data 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 on his own computers and 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. Finally, it is an open source project, inviting scientists to extend it in ways most useful to them.

OpenML offers various services to share and find datasets, to download or create scientific *tasks*, to share and find algorithms (called *flows*), and to share and organize results. These services are available through the OpenML website, as well as through a REST API for integration with software tools.

4.4.1 Datasets

Anyone can provide the community with new datasets to analyse. To be able to analyse the data, OpenML accepts a limited number of formats. For instance, currently it requires the ARFF format [61] for tabular data, although more formats will be added over time.

Table 4.1: Standard Meta-features.

Category	Meta-features
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 [108]	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 data can either be uploaded or 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 [146]. In some cases, such as Twitter feeds, data may be dynamic, which means that results won't be repeatable. However, in such tasks, repeatability is not expected. OpenML will automatically version each newly added dataset. Optionally, a user-defined version name can be added for reference. Next, authors can state how the data should be attributed, and which (creative commons) licence they wish to attach to it. Authors can also add a reference for citation, and a link to a paper. Finally, extra information can be added, such as the (default) target attribute(s) in labelled data, or the row-id attribute for data where instances are named.

For known data formats, OpenML will then compute an array of data characteristics, also called *meta-features*. Typical meta-features are often categorized as either simple, statistical, information theoretic or landmarks. Table 4.1 shows some meta-features computed by OpenML.

OpenML indexes all datasets and allows them to be searched through a standard keyword search and search filters. Each dataset has its own page with all known information. This includes the general description, attribution information, and data characteristics, but also statistics of the data distribution and, and all scientific *tasks* defined on this data (see below). It also includes a discussion section where the data-

set and results can be discussed.

4.4.2 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, labelled input data, and require predictions as outputs.

OpenML covers the following task types:

Supervised Classification Given a dataset with a nominal target and a set of train/test splits (e.g., generated by a cross-validation procedure) train a model and return the predictions of that model. The server will evaluate these, and compute standard evaluation measures, such as predictive accuracy, f-measure and area under the ROC curve.

Supervised Regression Given a dataset with a numeric target and a set of train/test splits (e.g., generated by a cross-validation procedure) train a model and return the predictions of that model. The server will evaluate these, and compute standard evaluation measures, such as root mean squared error (RMSE), mean absolute error and root relative squared error.

Learning Curve Analysis A variation of Supervised Classification. Given a dataset with a nominal target, various data samples of increasing size are defined. A model is built for each individual data sample. For each of these samples, various evaluation measures are calculated; from these a learning curve can be drawn. Chapter 6 will elaborate on this.

Data Stream Classification The online version of classification. Given a sequence of observations, build a model that is able to process these one by one and adapts to possible changes in the input space. Chapter 5 will elaborate on this.

Machine Learning Challenge A variation of Supervised Classification, similar to the setup of Kaggle [29]. The user is presented with a partly labelled dataset. The task is to label the unlabelled instances. As a result, there can be no cross-validation procedure, as there will always be a completely hidden test set.

Subgroup Discovery Given a dataset, return a conjunction of rules that describes a subgroup that is interesting with respect to a given quality measure. These quality measures can be weighted relative accuracy, jaccard measure or chi-squared.

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)

Figure 4.1: Example of an OpenML task description.

4.4.3 Tasks

If scientists want to perform, for instance, classification on a given dataset, they can create a new machine learning *task*. Tasks are instantiations of task types with specific inputs (e.g., datasets). Tasks are created once, and then downloaded and solved by anyone.





An example of such a task is shown in Figure 4.1. In this case, it is a classification task defined on dataset ‘anneal’ (version 1). Next to the dataset, the task includes the target attribute, the evaluation procedure (here: 10-fold cross-validation) and a file with the data splits for cross-validation. The latter ensures that results from different researchers can be objectively compared. For researchers doing an (internal) hyperparameter optimization, it also states the evaluation measure to optimize for. The required outputs for this task are the predictions for all test instances, and optionally, the models built and evaluations calculated by the user. However, OpenML will also compute a large range of evaluation measures on the server to ensure objective comparison.

Finally, each task has its own numeric id, a machine-readable XML description, as well as its own web page including all runs uploaded for that task and leaderboards.

4.4.4 Flows

Flows are implementations of single algorithms, workflows, or scripts designed to solve a given task. Flows can either be uploaded directly (source code or binary)

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

Figure 4.2: Example of an OpenML flow.

or reference it by URL. The latter is especially useful if the code is hosted on an open source platform such as GitHub or CRAN. Flows can be updated as often as needed. OpenML will version each uploaded flow, while users can provide their own version name for reference. Ideally, what is uploaded is software that takes a task id as input and then produces the required outputs. This can be a wrapper around a more general implementation. If not, the description should include instructions detailing how users can run an OpenML task (e.g., to verify submitted results).

OpenML stores meta-data about the uploaded flow, such as the dependencies, a

flow description and citation information. A flow can also contain one or more parameters. For each parameter, the name, description and default value (if known) are stored. Flows can also contain subflows. This way meta-algorithms such as Bagging can be distinguishable, e.g., Bagging CART trees would be different from Bagging REP Trees.

It is also possible to annotate flows with characteristics (similar to the characteristics in Table 3.1 on page 36), such as whether it can handle missing attributes, (non)numeric features and (non)numeric targets. 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, as well as a discussion section, see Figure 4.2.

It is important to emphasize that, although flows can contain pieces of software, these are not executed on the OpenML server. Flows are executed on the computer(s) of the users; it is their responsibility to link the uploaded results to the correct flow. In that sense, a flow on the OpenML server is a reference that links all the results obtained by the same algorithm to each other.

4.4.5 Setups

A setup is the combination of a flow and a certain setting of the parameters. Whenever a flow is uploaded, the user can also register the parameters that exist. It has been noted that parameter settings have a tremendous effect on the performance of an algorithm [9]. This widely accepted claim is easy to confirm using the experimental results of OpenML and the concepts of setups, as we will see further on. Setups that are run with default parameter settings, are flagged as such.

4.4.6 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 4.3. 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 the evaluations per fold. The fold-specific scores are aggregated as standard deviations in the web-interface, but can be obtained individually via the Rest API. For class-specific measures such as area under the ROC curve, precision and recall, per-class results are stored. Also the confusion matrix is available. Apart from the shown evaluation measures, a wide range of other evaluation measures is

also calculated, e.g., f-measure, kappa and root mean squared error. Additional information, such as run times and details on hardware can be provided by the user.

Moreover, because each run is linked to a specific task, flow, setup, and author, OpenML can aggregate and visualize results accordingly.

4.4.7 Studies

Studies are a combinations of datasets, tasks, flows, setups and runs. It is possible to link all these resources together, resulting in a permanent link that can be referred to in papers. Studies have a web-interface where general information can be given and results can be discussed. Having this all linked together makes it convenient for journal reviewers to verify the obtained results, for fellow researchers to build upon each others results and for anyone in general to investigate earlier obtained results.

4.4.8 Plug-ins

OpenML is being integrated in several popular machine learning environments, so that it can be used out of the box. These *plug-ins* can be downloaded from the web-site. Figure 4.4 shows how OpenML is integrated in WEKA's Experimenter [61]. 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 many additional features, such as the use of filters (for pre-processing operations), uploading of parameter sweep traces (for parameter optimization) and uploading of human readable model representations.

It has been widely recognized that the quality of an algorithm can be markedly improved by also selecting the right pre-processing and post-processing operators [41, 95, 145]. For example, the quality of k Nearest Neighbour algorithms typically degrades when the number of features increases [50], so it makes sense to combine these algorithms with feature selection [77, 111] or feature construction. The complete chain of pre-processing operators, algorithms and post-processing operators is typically referred to as a *workflow*. In order to extend the support for workflow research, OpenML is integrated in RapidMiner [121]. The integration consists of three new RapidMiner operators: one for downloading OpenML tasks, one for executing them and one for uploading the results, see Figure 4.5.

Typically, they will be connected as shown in Figure 4.5a. However, this modularization in three operators will likely be beneficial in some cases. The operators require an OpenML account to interact with the server. The "Execute OpenML Task" is a so-called *super-operator*; it contains a sub-workflow that is expected to solve the task that is delivered at the input port. The subroutine is executed for each defined





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

Evaluations

	0.9565 ± 0.0516			
Area under the roc curve	Iris-setosa	Iris-versicolor	Iris-virginica	
	0.98	0.9408	0.9488	
	actual\predicted	Iris-setosa	Iris-versicolor	Iris-virginica
Confusion matrix	Iris-setosa	48	2	0
	Iris-versicolor	0	47	3
	Iris-virginica	0	3	47
	0.9479 ± 0.0496			
Precision	Iris-setosa	Iris-versicolor	Iris-virginica	
	1	0.9038	0.94	
Predictive accuracy	0.9467 ± 0.0653			
	0.9467 ± 0.0653			
Recall	Iris-setosa	Iris-versicolor	Iris-virginica	
	0.96	0.94	0.94	

Figure 4.3: Example of an OpenML run.

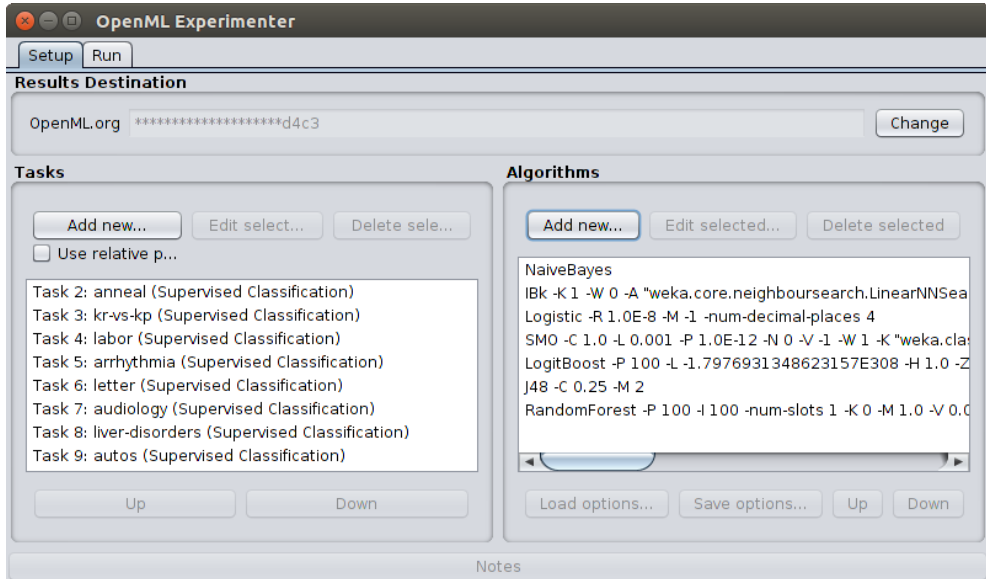
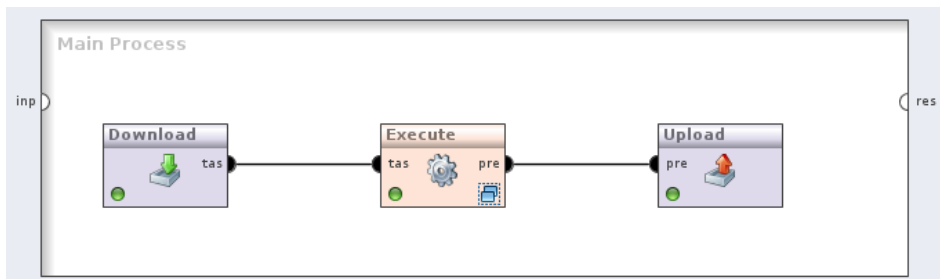


Figure 4.4: WEKA integration of OpenML.

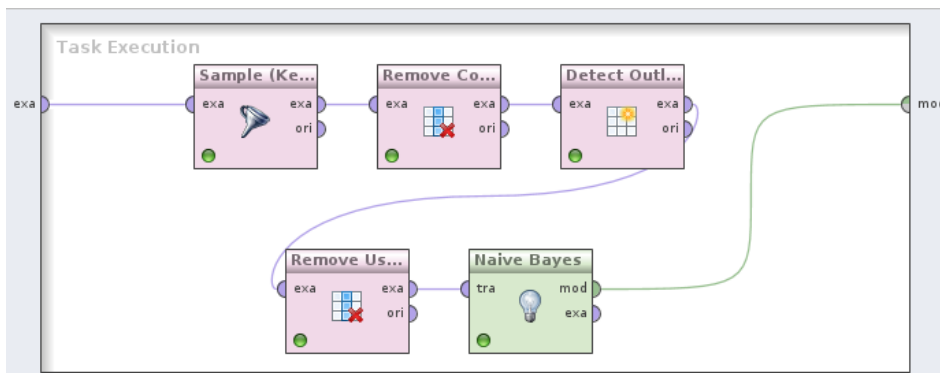
training set, and produces a model. This model is then used to predict the labels for the observations in the associated test set. An example of such a sub-workflow, including several pre-processing steps, is shown in Figure 4.5b. The output of this super-operator is a data structure containing predictions for all instances in the test sets, and basic measurements such as run times.

Additionally, researchers that use R can use the *openml* package, to work in compliance with the *mlr* package [16]. It supports a wide range of functionalities, mainly focussing on supervised classification and regression. An example of how to run an OpenML task is shown in Figure 4.6. OpenML is also integrated in ‘Scikit-learn’ [28, 103], a common Python package for Machine Learning. It supports similar functionalities as the *openml* package in R.

Furthermore, OpenML is integrated in MOA [13]. It has been widely recognized that mining data streams differs from conventional batch data mining [14, 118]. In the conventional batch setting, usually a limited amount of data is provided and the goal is to build a model that fits the data as well as possible, whereas in the data stream setting, there is a possibly infinite amount of data, with concepts possibly changing over time, and the goal is to build a model that captures general trends. With the MOA plug-in, OpenML facilitates data stream research, as we will see in Chapter 5.



(a) Main Workflow



(b) Subroutine solving OpenML task

Figure 4.5: Example of a RapidMiner workflow solving an OpenML task.

Finally, OpenML is also integrated in Cortana [92], a workbench that facilitates Subgroup Discovery [7]. Subgroup Discovery is a form of Supervised Learning that aims to describe certain parts in the data that comply to a certain measure of interestingness. What quality measure is desired, is defined in the OpenML task.

All-together, these plug-ins enable frictionless collaboration, as users do not have any additional burden to share their experimental results. This leads to an extensive database of experiments, that enables us to learn from the past.

4.5 Learning from the past

Having a vast set of experiments, collected and organized in a structured way, allows us to conduct various types of experiments. In this chapter, we will demonstrate various ways of research that OpenML facilitates. Vanschoren et al. [153] describes specifically three types of experiments, offering increasingly generalizable results:

```

1 library(mlr)
2 library(OpenML)
3
4 # set API key to read only key (replace it with your own key)
5 setOMLConfig(apikey = "b2994bdb7ecb3c6f3c8f3b35f4b47f1f")
6
7 lrn = makeLearner("classif.randomForest")
8
9 # upload the new flow (with information about the algorithm and settings);
10 # if this algorithm already exists on the server, one will receive a message
11 # with the ID of the existing flow
12 flow.id = uploadOMLFlow(lrn)
13
14 # the last step is to perform a run and upload the results
15 run.mlr = runTaskMLr(task, lrn)
16 run.id = uploadOMLRun(run.mlr)

```

Figure 4.6: R integration of OpenML.

Model-level analysis evaluate machine learning methods over one or multiple datasets, using a given performance measure (e.g., predictive accuracy or area under the ROC curve). These studies give insight in HOW a particular method works.

Data-level analysis give insight in how the performance of specific machine learning methods is affected by given data characteristics (e.g., number of features, number of classes). These studies attempt to explain WHEN (on which kinds of data) a particular method should be preferred over the other.

Method-level analysis leverage given algorithm characteristics (e.g., bias-variance profile, model predictions on earlier datasets) in order to explain WHY a particular algorithm behaves the way it does.

4.5.1 Model-level analysis

In this first type of study, we use the vast amount of runs in OpenML to gain insight in how certain algorithms perform. Many flows (and setups) are run over a vast set of tasks, and the results can be used to benchmark, compare or rank algorithms.

4.5.1.1 Comparing flows

For each task, OpenML automatically plots all evaluation scores of all performed algorithms. Figure 4.7 shows this for the commonly used UCI dataset ‘letter’ [47]. In

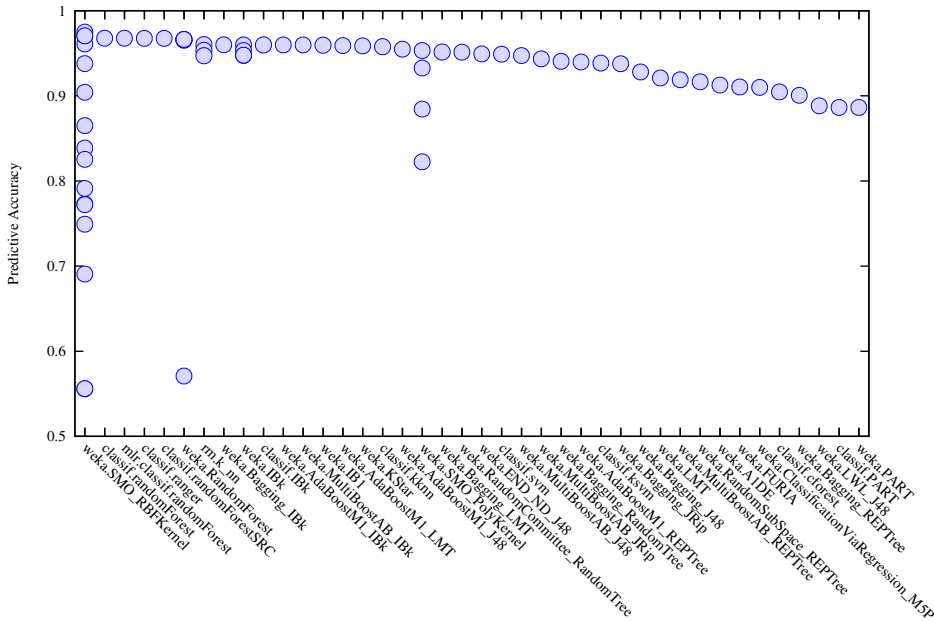


Figure 4.7: Performance of various algorithms on the 'letter' dataset.

this dataset the task is to classify letters produced by various fonts, based on pre-extracted attributes. This dataset has 26 classes (each letter from the alphabet is a class), which is fairly high for a classification task. The x -axis shows a particular flow, and each dot represents a given parameter setting that obtained a certain score. This type of query shows what kind of algorithm is suitable for a given dataset.

In order not to overload the image, we only show the 40 best performing flows. Ensemble methods are grouped by the base-learner that is used (e.g., Bagging IBk is considered a different flow than Bagging J48). The result contains flows from various classification workbenches that are integrated in OpenML, i.e., Weka, R and Rapid-Miner. In this case, it seems that instance-based methods perform fairly well. Among the top performing algorithms, there are many variations of Random Forest and k -NN, with IBk being the Weka version of instance-based classification.

A similar image was published in [153], based on the experiment database for machine learning. The results are similar to the ones we present. However, the results from OpenML are based on more algorithms from more toolboxes, produced by various people from all over the world. For this reason, we expect that the observations

Table 4.2: Classifiers and the important parameters that were optimized for populating the database.

Classifier	Parameters	Range	Scale
SVM (RBF Kernel)	complexity	$[2^{-12}-2^{12}]$	log
	gamma	$[2^{-12}-2^{12}]$	log
J48	confidence factor	$[10^{-4}-10^{-1}]$	log
	minimal leaf size	$[2^0-2^6]$	log
<i>k</i> -NN	number of neighbours	[1-50]	linear
Logistic Regression	ridge	$[2^{-12}-2^{12}]$	log
Random Forest	nr of variables per split	$[2^1-2^8]$	log
LogitBoost (REPTree)	shrinkage	$[10^{-4}-10^{-1}]$	log
	max depth	[1-5]	linear
	number of iterations	[500-1000]	linear

made upon OpenML experiments are more substantial.

4.5.1.2 Effect of parameter optimization

OpenML facilitates that parameter optimization techniques can store all intermediate results, giving more options to analyse the specific effects of parameters. We used Weka's MultiSearch package to populate the database with the classifiers and parameter settings specified in Table 4.2. If the number of parameter combinations exceeded 200, Random Search [8] was used to randomly select 200 parameter settings.

A 10-fold cross-validation procedure was used, with for each fold an internal 2-fold cross-validation procedure to select the best parameter setting for this fold, resulting in at most 2,000 attempted setups per run. Figure 4.8 shows violin plots of the varying accuracy results. All individual results can be obtained from OpenML.

These kind of studies lead to interesting observations. They show that especially Support Vector Machines need proper parameter tuning: the median performance is very low compared to the maximum performance. The probability of obtaining a low score for this is rather high, as can be seen by the high density area at the bottom of the plot. The other algorithms perform more robust, especially Random Forest and Decision Tree (high density at the top of the plot, small tail at the bottom). By further inspecting the results, we observe that all outliers can be attributed to a sub-optimal value of a specific parameter. The outliers of the J48 decision tree were all produced by a high value (64) for the parameter that determines the minimal leaf size. The outliers of the Random Forest were produced by a low value (2) for the attributes that are available at each split. The outliers of LogitBoost were produced when the

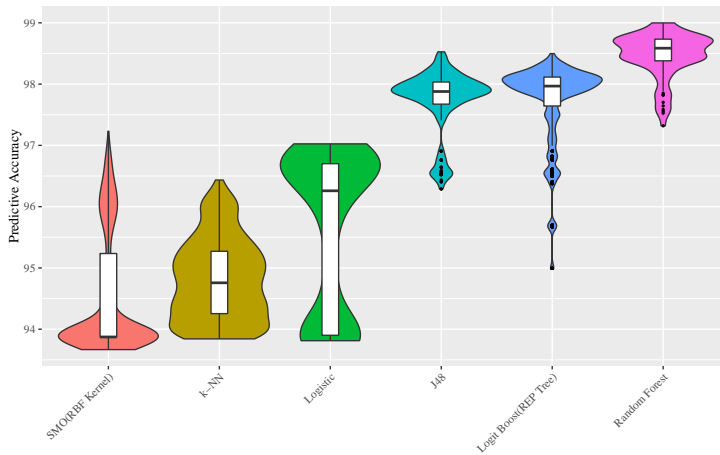


Figure 4.8: Variation in predictive accuracy when optimizing parameters on the “sick” dataset.

base-learner was built with a maximum depth of 1, effectively making it a decision stump.

It is plausible that when using a decision tree, a relation exists between the number of instances and the optimal value for minimal leaf size. If the dataset is too small, setting this value too high restricts the flexibility of the model, possibly leading to under-fitted models. Similarly, when building a Random Forest, having too few attributes to select a split from can lead to suboptimal trees. Although these results will make sense to most machine learning experts, currently there are no publications or data to back up these observations. These kind of data-driven experiments can be a first step towards a better understanding of parameter behaviour in machine learning.

4.5.1.3 Parameter effect across datasets

It is possible to track the effect of a certain parameter over a range of values. Figure 4.9 plots the effect of the gamma parameter of the RBF kernel for Support Vector Machines, on various datasets.

As we can see, the parameter has a similar effect on most of the displayed datasets. By increasing the value, performance grows to an optimum. After that it rapidly degrades. In the case of ‘car’, ‘optdigits’ and ‘waveform’, performance degrades to the default accuracy, after which it stabilizes. In the case of the letter dataset, the degradation hasn’t finished yet, but it is to be expected that it continues in a similar way. For the ‘soybean’ dataset, it stabilizes above this level. The optimal value of the

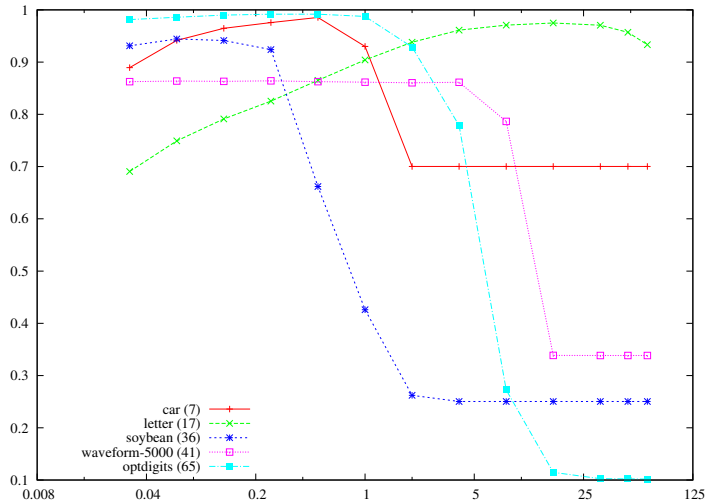


Figure 4.9: Effect of gamma parameter of the RBF kernel for Support Vector Machines on various datasets. The number between brackets indicates the number of attributes of such dataset.

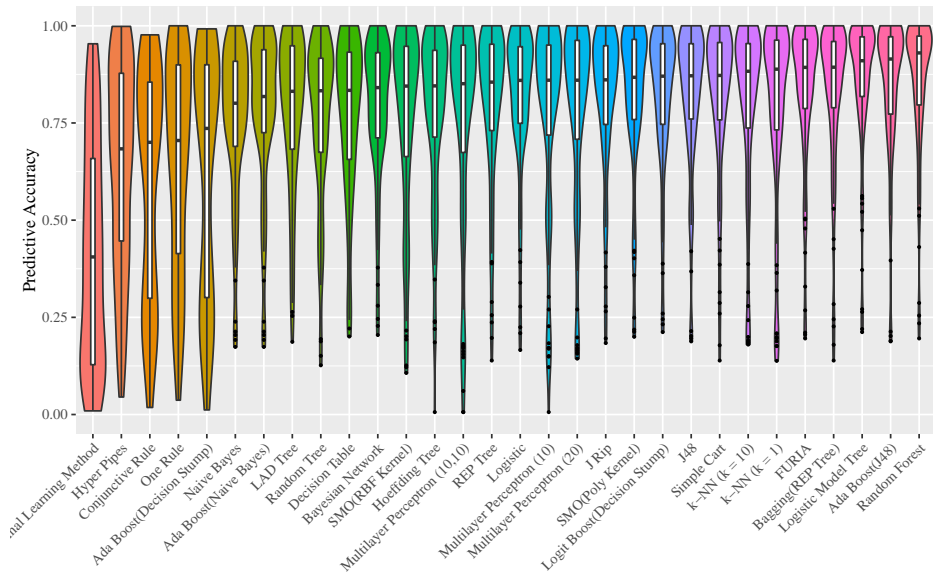
parameter is different for all datasets, as would be expected. It seems that setting this value too low is less harmful for performance than setting it too high. All-together, the parameter landscape seems to be smooth, i.e., there are no spikes in the plots.

4.5.1.4 Comparison over many datasets

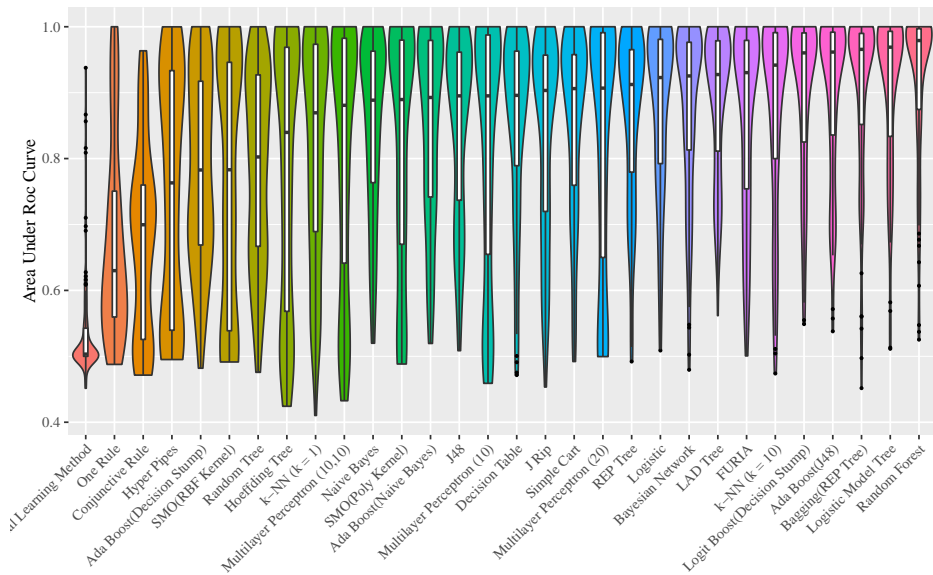
In order to gain a decent understanding of how a classifier performs, it could be evaluated over a wide range of datasets. We selected a set of classifiers and datasets from OpenML. All classifiers were ran on all datasets.

Figure 4.10 shows violin plots of various algorithms over a well selected set of datasets (complete list in Table 6.1 on page 115). Violin plots show the probability density of the performance scores at different values [66]. Additionally, a box plot is shown in the middle. The classifiers are sorted based on the median. Classifiers to the right perform generally better than classifiers to the left. Random Forest [24] performs best on average, but also the other ensembles from Chapter 3 perform good, e.g., Adaptive Boosting [46] and Logistic Boosting [48]. Logistic Model Tree (LMT) [84] (which is a combination of trees and Logistic Regression) also performs reasonably well.

Note that when a classifier is ranked low, it does not necessarily mean that is it a



(a) Accuracy



(b) Area under the ROC Curve

Figure 4.10: Ranking of algorithms over a set of 105 datasets.

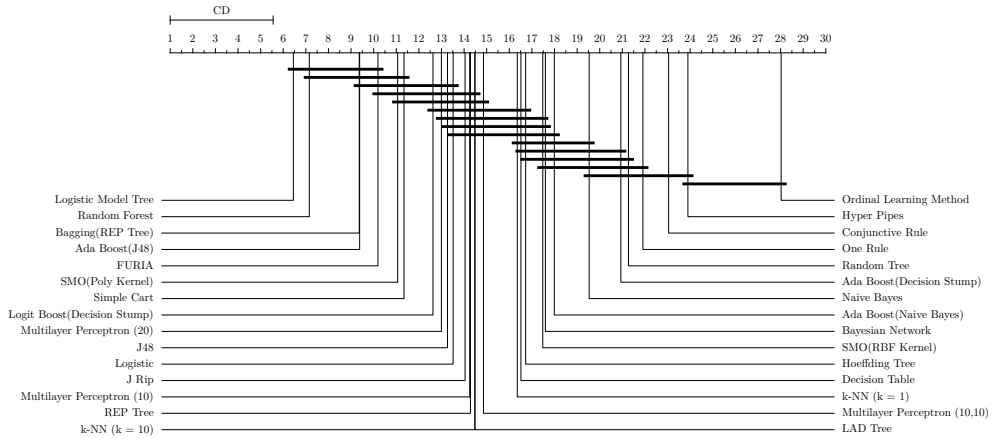


Figure 4.11: Results of Nemenyi test ($\alpha = 0.05$) on the predictive accuracy of classifiers in OpenML. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

bad classifier. For example, Naive Bayes does not score well on the general ranking, but is used quite often in text mining, which suggests that it is suitable for that specific task. When a classifier does not end high in the general ranking, this typically indicates that the algorithm designer must put more effort in specifying for what types of data it works well.

To assess statistical significance of such results, we can use the Friedman test with post-hoc Nemenyi test to establish the statistical relevance of our results. These tests are considered the standard when it comes to comparing multiple classifiers [36]. The Friedman test checks whether there is a statistical significant difference between the classifiers; when this is the case the Nemenyi post-hoc test can be used to determine which classifiers are significantly better than others. A statistical test operates on a given evaluation measure.

Figure 4.11 shows the result of the Nemenyi test on the predictive accuracy of the classifiers from Figure 4.10. We see a similar order of classifiers as in Figure 4.10; again the Logistic Model Tree and Random Forest perform best. Classifiers that are connected by a horizontal line are statistically equivalent, e.g., there was no statistical evidence that the Logistic Model Tree is better than the SVM with Polynomial kernel.

These results are obtained by running the respective algorithms on the datasets without hyperparameter tuning. It has been noted that hyperparameter optimization has a big influence on the performance of algorithms. It would be interesting to see how an optimization procedure (e.g., Random Search) will affect these rankings.

4.5.2 Data-level analysis

In the previous chapter, we used the experiments from OpenML to gain insight in how algorithms perform. Meta-learning focusses on when algorithms are expected to perform well. In this chapter we demonstrate how to obtain this knowledge.

4.5.2.1 Data property effect

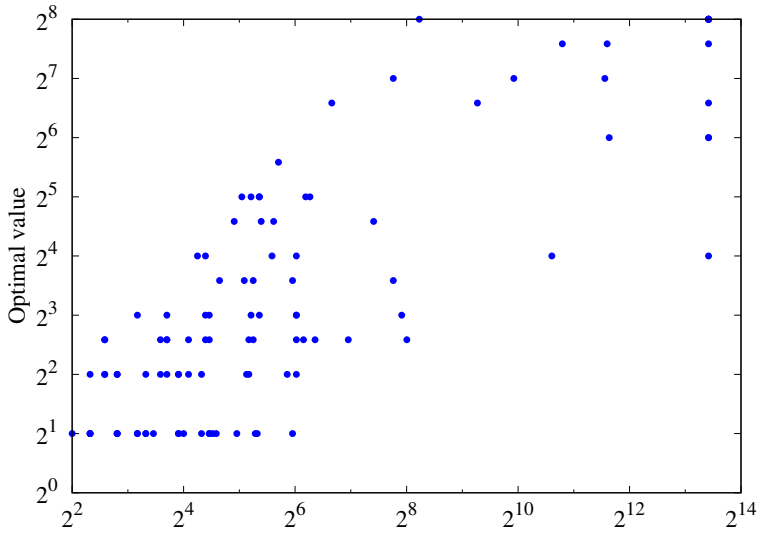
As OpenML contains many runs of algorithms with different parameter settings, we can use these results to gain more insight in the interplay between a certain data characteristic and the optimal value of such parameter. In order to do so, we fix all parameters to a given value (e.g., the default value) and vary the one that we are interested in. The optimal value is the value with which the algorithm obtains the highest performance on a given performance measure; in this case predictive accuracy. In case of a tie, a lower value was preferred.

The Random Forest classifier has many parameters. In order to assure more diversity in the individual trees, at each node the tree induction algorithm can only select a splitting criteria out of a restricted set of randomly chosen attributes. The ‘Number of Features’ parameter controls the number of attributes that can be chosen from. Intuitively, there is a relationship between the number of features of the dataset and the optimal value for this parameter. This intuition is built upon in popular machine learning workbenches. For example, in Weka [61] this parameter defaults to the logarithm of the number of features, and in Scikit-learn this value defaults to the square root of the number of features. Figure 4.12a plots the optimal value against the ‘Number of Features’ data characteristic. Figure 4.12b plots the optimal value against the ‘Dimensionality’ data characteristic (which is the number of features divided by the number of instances). By definition, there can be no optimal values in the top left area of Figure 4.12a: the optimal value for this parameter can not be higher than the actual number of features of a dataset.

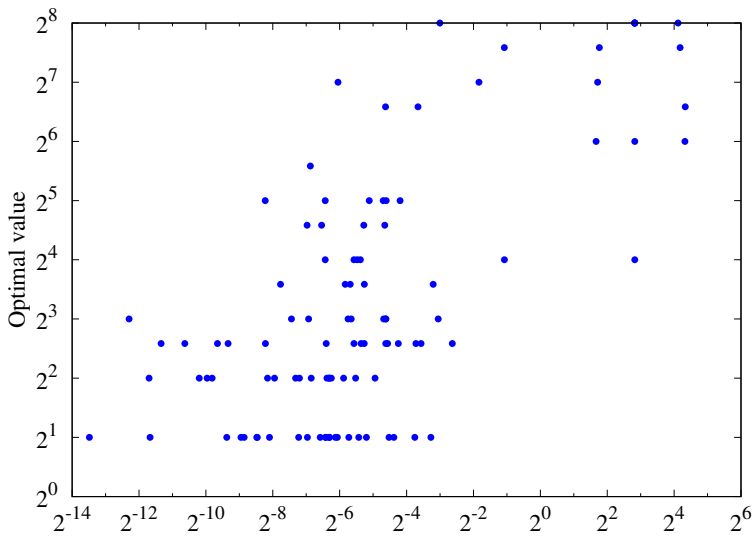
The plot seems to confirm some sort of relation between the number of attributes and the optimal value for this parameter. However, these scatter-plots also have some intrinsic disadvantages: overlapping points are not visible as such, it only shows the absolute best value of the parameter and the results are circumstantial (i.e., the results can be completely different when other parameters are varied). Conclusions should be drawn with great care.

4.5.2.2 Effect of feature selection

It is often claimed that data pre-processing is an important factor contributing towards the performance of classification algorithms. We can use the experiments in



(a) Number of Features



(b) Dimensionality

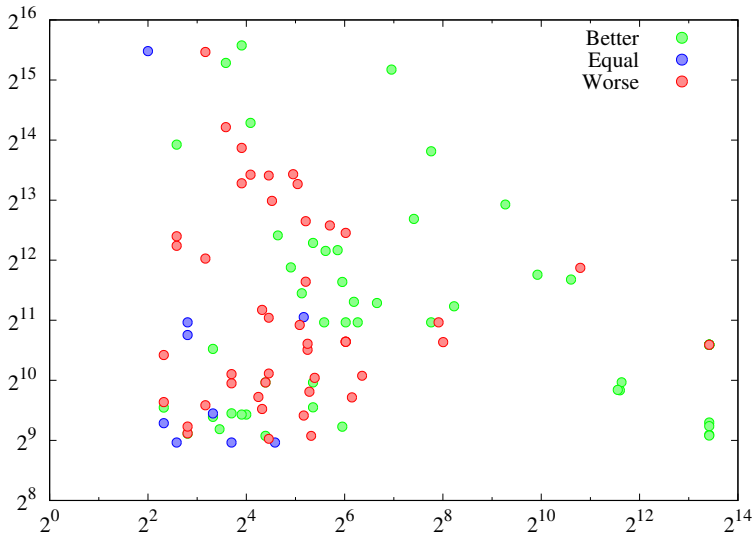
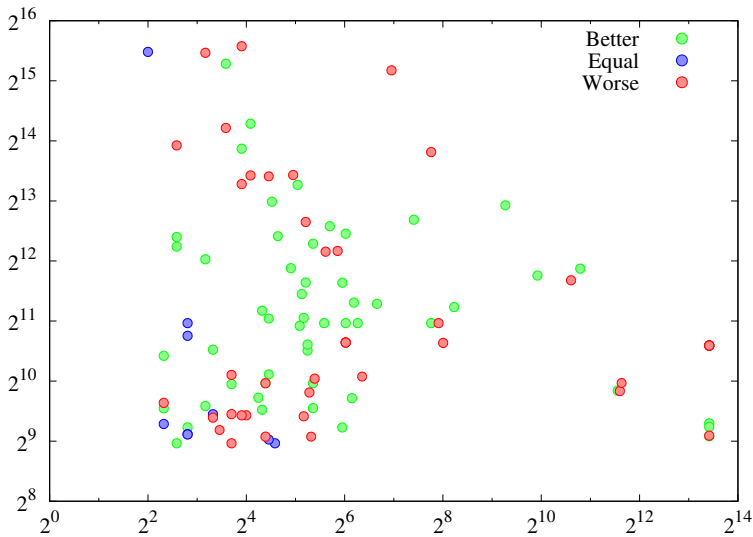
Figure 4.12: The optimal values of the 'Random Features per Split' parameter of Random Forests

OpenML to investigate whether this is true, and on what kind of data this is the case. We will focus on feature selection. Real world data sets can be rife with irrelevant features, especially if the data was not gathered specifically for the classification task at hand. For instance, in many business applications hundreds of customer attributes may have been captured in some central data store, whilst only later is decided what kind of models actually need to be built [114]. In order to help classifiers building a good model, a feature selection procedure can be adopted, selecting a representative set of features.

Many OpenML tasks have for a given classifier results on how it performed with and without various pre-processing operators. For example, in Weka, we can use the Feature Selected Classifier, to apply feature selection on a given dataset. We simply combine the results of that algorithm with and without the feature selection procedure, and store which one performed better. There are many different feature selection techniques. In this experiment, we used Correlation-based Feature Subset Selection as feature selector. It attempts to identify features that are highly correlated with the target attribute, yet uncorrelated with each other [62]. However, also many other feature selection procedures exists.

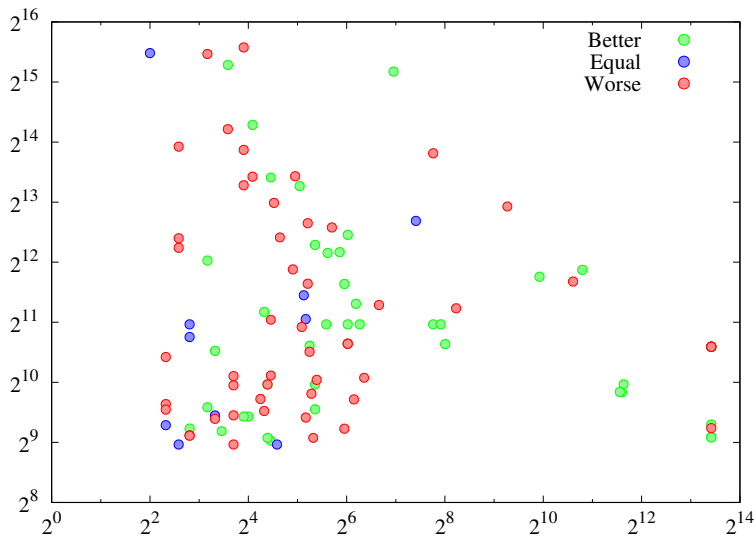
We can plot the effect of feature selection on classifier performance against data characteristics. In Figure 4.13, each dot represents a dataset. Its position on the x -axis represents the number of attributes of that dataset, and its position on the y -axis represents the number of instances of that dataset. Then the colour of the dot shows whether feature selection yielded better or worse results.

These scatter-plots show both some expected behaviour as well as some interesting patterns. First of all, we can see that feature selection is most beneficial for methods such as k -NN (Figure 4.13a) and Naive Bayes (Figure 4.13b). This is exactly what we would expect: due to the curse of dimensionality, nearest neighbour methods can suffer from too many attributes [117] and Naive Bayes is vulnerable to correlated features [78]. Quite naturally, the trend seems that when using k -NN, feature selection yields good results on datasets with many features [111]. We also see unexpected behaviour. For example, it has been noted that some tree-induction algorithms have built-in protection against irrelevant features [115]. However, it can be observed that still in many cases it benefits from feature selection (Figure 4.13c). Also, as one of the most dynamic and powerful model types, MultiLayer Perceptrons are considered to be capable of selecting relevant features (Figure 4.13d). However, in order to do so, the parameters need to be tuned accordingly. Intuitively, the more features the dataset contains, the more epochs are needed to learn a good model. If there is not enough budget to invest in an appropriate number of epochs, feature selection can serve as an alternative.

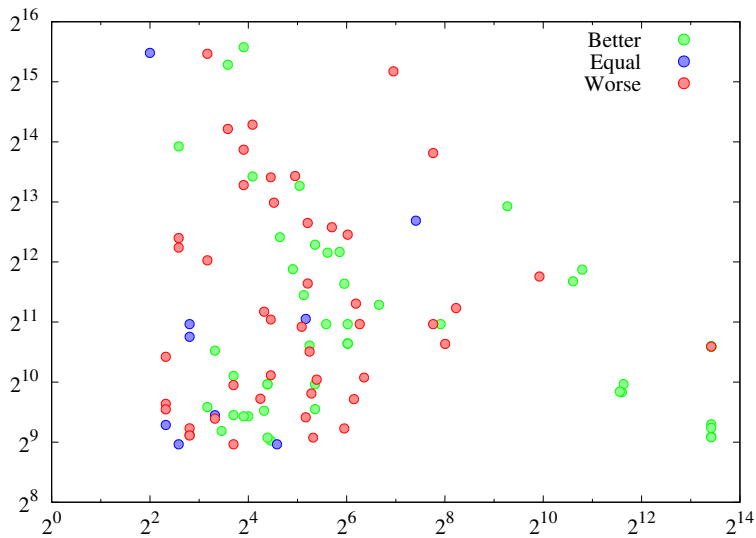
(a) k -NN

(b) Naive Bayes

Figure 4.13: The effect of feature selection on classifier performance, plotted against two data characteristics (number of features on the x -axis, number of instances on the y -axis). Each dot represents a dataset, the colour indicates whether the performance was increased or decreased by feature selection.



(c) Decision Trees



(d) Multilayer Perceptron

Figure 4.13: The effect of feature selection on classifier performance, plotted against two data characteristics (number of features on the x -axis, number of instances on the y -axis). Each dot represents a dataset, the colour indicates whether the performance was increased or decreased by feature selection (continued).

4.5.3 Method-level analysis

In this last type of experiment, we will use the experimental results in OpenML to generalize over methods. This way we attempt to gain more insight in why these algorithms behave the way they do.

4.5.3.1 Instance-based analysis

For many tasks, OpenML stores the individual predictions that are done by specific algorithms. We can use these to create a hierarchical agglomerative clustering of data stream classifiers in an identical way to the authors of [85]. Classifier Output Difference is a metric that measures the difference in predictions between a pair of classifiers. For each pair of classifiers, we use the number of observations for which the classifiers have different outputs, aggregated over all data streams involved. Hierarchical agglomerative clustering (HAC) converts this information into a hierarchical clustering. It starts by assigning each observation to its own cluster, and greedily joins the two clusters with the smallest distance [132]. The complete linkage strategy is used to measure the distance between two clusters. Formally, the distance between two clusters A and B is defined as $\max \{COD(a, b) : a \in A, b \in B\}$.

Figure 4.14 shows the resulting dendrogram, built over a large set of data stream classifiers provided by MOA. Although the specific details of data stream classification will be explained in Chapter 5, we can already make some observations. The figure confirms some well-established assumptions. The clustering seems to respect the taxonomy of classifiers provided by MOA. Many of the tree-based and rule-based classifiers are grouped together. There is a cluster of instance-incremental tree classifiers (Hoeffding Tree, AS Hoeffding Tree, Hoeffding Option Tree and Hoeffding Adaptive Tree), a cluster of batch-incremental tree-based and rule-based classifiers (REP Tree, J48 and JRip) and a cluster of simple tree-based and rule-based classifiers (Decision Stumps and One Rule). Also the Logistic and SVM models seem to produce similar predictions, having a sub-cluster of batch-incremental classifiers (SMO and Logistic) and a sub-cluster of instance incremental classifiers (Stochastic Gradient Descent and SPegasos with both loss functions).

The dendrogram also provides some surprising results. For example, the instance-incremental Rule Classifier seems to be fairly distant from the tree-based classifiers. As decision rules and decision trees work with similar decision boundaries and can easily be translated to each other, a higher similarity would be expected [6]. Also the internal distances in the simple tree-based and rule-based classifiers seem rather high.

A possible explanation for this could be the mediocre performance of the Rule Classifier. Even though COD clusters are based on instance-level predictions rather than accuracy, well performing classifiers have a higher prior probability of being

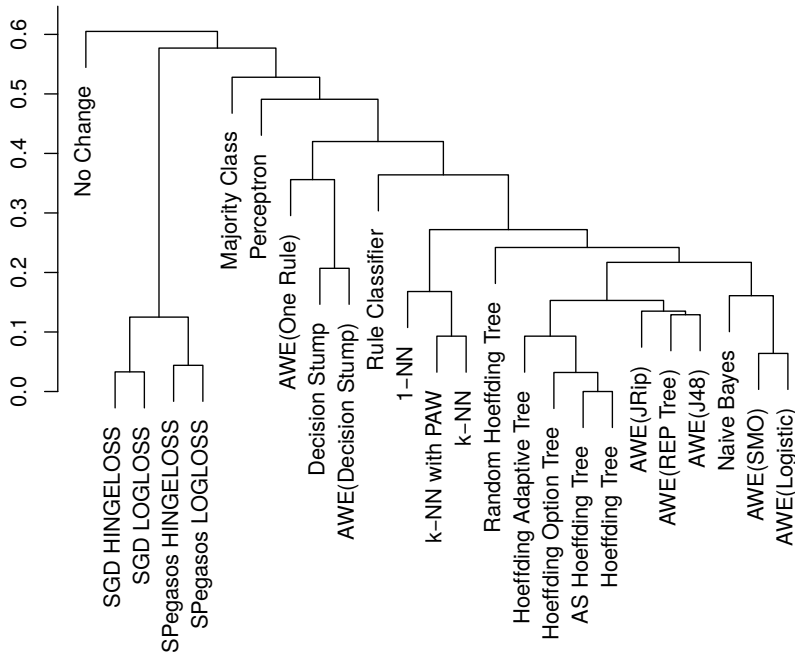


Figure 4.14: Hierarchical clustering of stream classifiers, averaged over 51 data streams from OpenML.

clustered together. As there are only few observations they predict incorrectly, naturally there are also few observations their predictions can disagree on.

4.6 Conclusions

In many sciences, networked science tools are allowing scientists to make discoveries much faster than was ever possible before. Hundreds of scientists are collaborating to tackle hard problems, individual scientists are building directly on the observations of all others, and students and citizen scientists are effectively contributing to real science.

To bring these same benefits to machine learning researchers, we introduced OpenML, an online service to share, organize and reuse data, code and experiments. Following best practices observed in other sciences, OpenML allows collaborations to scale effortlessly and rewards scientists for sharing their data more openly.

We have shown various types of studies that can be done with the results in OpenML. Apart from many new discoveries that can be done, these studies also con-

firm or disclaim well established assumptions in a data-driven way.

We believe that this new, networked approach to machine learning will allow scientists to work more productively, make new discoveries faster, be more visible, forge many new collaborations, and start new types of studies that were practically impossible before.