Massively Collaborative Machine Learning

Jan N. van Rijn

The author of this PhD thesis was employed at Leiden University, and also used facilities of the University of Waikato.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

The author was funded by the Netherlands Organization for Scientific Research (NWO) as part of the project 'Massively Collaborative Data Mining' (number 612.001.206).

Typeset using LaTeX, diagrams generated using GGPLOT and GNUPLOT
Cover image by Olivier H. Beauchesne and SCImago Lab (used with permission)
Printed by Ridderprint B.V.
ISBN 978-94-6299-506-2

Massively Collaborative Machine Learning

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op maandag 19 december 2016
klokke 12:30 uur

door

Jan Nicolaas van Rijn
geboren te Katwijk
in 1987

Promotiecommissie

Promotor:	prof. dr. J. N. Kok
Copromotores:	dr. A. J. Knobbe
	dr. J. Vanschoren (Technische Universiteit Eindhoven)
Promotiecommissie:	prof. dr. T. H. W. Bäck (secretaris)
	prof. dr. E. Marchiori (Radboud Universiteit)
	prof. dr. B. Pfahringer (University of Waikato, Nieuw Zeeland)
	prof. dr. A. Plaat (voorzitter)

Contents

List of Figures

1 Introduction

1.1 Introduction

We are surrounded by data. On a daily basis, we are confronted by many forms of it. Companies try to spread their commercials by means of billboards, commercials and online advertisements. We have instant access to our friends' social lives using services as Facebook and Twitter, and we can obtain information about countless topics of interest by means of websites such as Wikipedia. In most cases, this is a double-edged sword. Companies and governments also collect information about us. For example, most websites store information about our browsing behaviour, banks know most about our financial transactions, and telecom providers even have access to our exact whereabouts, as our GPS coordinates are shared by our mobile phones.

Data is also gathered for scientific purposes. Large sensor networks and telescopes measure complex processes, happening around us on Earth or throughout the Universe. Any estimation of the amount of data that is being produced, transferred and gathered would be pointless, as it will be outdated some moments after publication.

All this data is valuable for the information, knowledge and eventually wisdom we could obtain from it. We could identify fraudulent transactions based on financial data, develop new medicines based on clinical data, or locate extraterrestrial life based on telescope data. This process is called *learning*. The scientific community has created many techniques for analysing and processing data. A traditional scientific tasks is modelling, where the aim is to describe the data in a simplified way, in order to learn something from it. Many data modelling techniques have been developed, based on various intuitions and assumptions. This area of research is called *Machine Learning*.

However, all data is different. For example, data about clinical trials is typically

very sparse, but well-structured, whereas telescopes gather large amounts of data, albeit initially unstructured. We cannot assume that there is one algorithm that works for all sorts of data. Each algorithm has its own type of expertise. We have only little knowledge about which algorithms work well on what data.

The field of Machine Learning contains many challenging aspects. The data itself is often big, describing a complex concept. Algorithms are complex computer programs, containing many lines of code. In order to study the interplay between these two, we need data about the data and the algorithms. This data is called *meta-data*, and learning about the learning process itself is called *meta-learning*. It is possible to gain knowledge about the learning process when there is sufficient meta-data. Some effort has been devoted to building a large repository of this experimental data, called the 'open experiment database' [153]. It contains a large amount of publicly available Machine Learning results. This way, existing experimental data can be used to answer new research questions. Although this has proven extremely useful, there is still room for improvement. For example, sharing experiments was difficult: while all experimental data was accessible to the public, contributing new results towards the experiment database was only practically possible for a small circle of researchers. Furthermore, sensibly defining the types of meta-data that are being stored would expand the range of information and knowledge that can be obtained from the data. For example, storing all evaluation measures per cross-validation fold enables statistical analysis on the gathered data, and storing the individual predictions of the algorithms enables instance-level analysis. Our aim is to build upon the existing work of experiment databases, and demonstrate new opportunities for Machine Learning and meta-learning.

Our contributions are the following. We have developed an online, open experiment database, called 'OpenML'. This enables researchers to freely share their data, algorithms and empirical results. This significantly scales up the size of typical machine learning and meta-learning studies. Implementing algorithms and modelling data are both time-intensive tasks. Instead of setting up the experiments themselves, researchers can now simply look up the results by querying the database, covering a much larger set of experiments. OpenML automatically indexes and organizes the uploaded meta-data, allowing researchers to investigate common questions about the performance of algorithms, such as which parameters are important to tune, what the effect is of a given data property on the performance of algorithms, or which algorithm works best on a certain type of data.

We have demonstrated the effectiveness of this collaborative approach to meta-learning with two large-scale studies that were not practically feasible before. The first study covers the data stream setting, which contains some challenging real-world aspects: large amounts of data need to be processed at high speed and learned models

can become outdated. In this work, we created a novel approach that, while processing the stream, dynamically changes the modelling algorithm when another algorithm seemed more appropriate. The study covered 60 data streams, which to the best of our knowledge, is the largest meta-learning study in the data stream literature to date.

The second study covers a conventional meta-learning task, where the goal is to find an algorithm that adequately models the dataset. However, it is also important to find that algorithm as fast as possible. Indeed, whenever such an algorithm is recommended, it can be tested (e.g., using cross-validation) and if its performance is not sufficient, another one can be tried, but this can be a very slow process. This study showed that there are techniques that trade off performance and run time. If one is willing to settle for an algorithm that is almost as good as the absolute best algorithm for that dataset, the run time can be decreased by orders of magnitude.

The remainder of this thesis is organised as follows. Chapter 2 introduces some basic aspects about Machine Learning, and introduces some well-known model types. Chapter 2 surveys common meta-learning techniques. It approaches meta-learning both from a learning and a search perspective. Chapter 4 describes the online experiment database on which we collect experimental results. Chapter 5 describes the first study that demonstrates the use of OpenML for data streams. Chapter 6 describes a second study that shows how meta-learning techniques can trade off accuracy and run time. Chapter 7 concludes and points to future work.

1.2 Publications

The different chapters of this thesis are based on the following peer-reviewed publications:

- J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. OpenML: A Collaborative Science Platform. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–649. Springer, 2013 (Chapter 4)

- J. N. Van Rijn, V. Umaashankar, S. Fischer, B. Bischl, L. Torgo, B. Gao, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. A RapidMiner extension for Open Machine Learning. In *RapidMiner Community Meeting and Conference*, pages 59–70, 2013 (Chapter 4)

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Algorithm Selection on Data Streams. In *Discovery Science*, volume 8777 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2014 (Chapter 5)

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Towards Meta-learning over Data Streams. In J. Vanschoren, P. Brazdil, C. Soares, and L. Kotthoff, editors, *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, number 1201 in CEUR Workshop Proceedings, pages 37–38, Aachen, 2014 (Chapter 5)

- J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014 (Chapter 4)

- J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren. Fast Algorithm Selection using Learning Curves. In *Advances in Intelligent Data Analysis XIV*, pages 298–309. Springer, 2015 (Chapter 6)

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Case Study on Bagging Stable Classifiers for Data Streams. In *Proceedings of the 24th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2015)*, 6 pages, 2015 (Chapter 5)

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Having a Blast: Meta-Learning and Heterogeneous Ensembles for Data Streams. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 1003–1008. IEEE, 2015 (Chapter 5)

- J. N. van Rijn and J. Vanschoren. Sharing RapidMiner Workflows and Experiments with OpenML. In J. Vanschoren, P. Brazdil, C. Giraud-Carrier, and L. Kotthoff, editors, *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection (MetaSel)*, number 1455 in CEUR Workshop Proceedings, pages 93–103, Aachen, 2015 (Chapter 4)

- J. Vanschoren, J. N. van Rijn, and B. Bischl. Taking machine learning research online with OpenML. In *Proceedings of the 4th International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 1–4. JLMR.org, 2015 (Chapter 4)

A full list of publications by the author can be found on page 155 of this thesis.

2 Machine Learning

As argued in Chapter 1, there is a great amount of data being generated every day. Collecting data in itself is only useful if we are able to make sense out of it. A great challenge lies in analysing and modelling the collected data.

2.1 Introduction

Model building is a time-consuming task that has already been practised for a long time by many scientists. In the 16th century, Nicolaus Copernicus described a model that considered the sun to be the centre of the universe, rather than the Earth. The data that he based this model on were the observations he made of the sun and its orbiting planets. Although his model was not entirely accurate, it is widely accepted that his ideas led to great scientific breakthroughs in his time and thereafter. Many examples of scientific models based on data can be found, where data can be literally anything, ranging from measurements obtained through a microscope to observations obtained through a telescope.

Machine Learning is the field of research that focuses on the *automatic* building of *predictive* models from *collected data*, that can be evaluated using an *objective* performance measure. As the term 'automatic' indicates, it uses algorithms, aiming to keep the human expert out of the loop. Of course, there are also Machine Learning techniques that deliberately use domain experts' knowledge [157]. However, even those techniques are focused on automating the process, effectively taking over the workload and suggesting interesting patterns that would be too complex to distinguish otherwise. As the term 'predictive' indicates, the models should be capable of making predictions for yet unseen data. It is easy to perform well on the already seen data

Table 2.1: Random sample from the 'iris' dataset, as provided by [44].

sepal length	sepal width	petal length	petal width	class
5.1	3.7	1.5	0.4	setosa
5.1	3.3	1.7	0.5	setosa
6.2	2.8	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica
6.1	3.0	4.6	1.4	versicolor
4.7	3.2	1.3	0.2	setosa
4.4	3.2	1.3	0.2	setosa
6.5	2.8	4.6	1.5	versicolor
6.8	3.0	5.5	2.1	virginica
6.3	3.4	5.6	2.4	virginica
5.1	3.5	1.4	0.3	setosa
6.3	3.3	6.0	2.5	virginica
5.0	3.2	1.2	0.2	setosa
5.1	3.8	1.9	0.4	setosa
5.7	4.4	1.5	0.4	setosa

sepal length	sepal width	petal length	petal width	class
5.5	4.2	1.4	0.2	setosa
6.4	3.2	4.5	1.5	versicolor
7.1	3.0	5.9	2.1	virginica
5.1	3.5	1.4	0.2	setosa
5.5	3.5	1.3	0.2	setosa
5.6	2.8	4.9	2.0	virginica
6.3	2.5	4.9	1.5	versicolor
5.8	4.0	1.2	0.2	setosa
6.0	2.2	5.0	1.5	virginica
5.0	3.4	1.5	0.2	setosa
4.8	3.0	1.4	0.3	setosa
6.8	3.2	5.9	2.3	virginica
5.8	2.6	4.0	1.2	versicolor
5.7	2.9	4.2	1.3	versicolor
7.4	2.8	6.1	1.9	virginica

(just memorize it); Machine Learning is about *generalizing* beyond this. Typically, the models will be evaluated based on predictions made for unseen data. An evaluation criterion can be the percentage of correct predictions, but in some cases more subtle measures are required. This is the objective performance measure.

The main concepts of Machine Learning are *data*, *tasks*, *models*, *algorithms* and *evaluation measures*. Chapter 2.2 gives examples of common types of data(sets), Chapter 2.3 overviews some common tasks that will recur in this thesis. In Chapter 2.4, we review the most common model types and mention the algorithms used to build them. Chapter 2.5 discusses the various ways of evaluating such models. Chapter 2.6 concludes with a discussion about *meta-learning*.

2.2 Data

In this chapter we will explore some common datasets that can be modelled using Machine Learning techniques.

2.2.1 Iris

Iris is a species of flowering plants, named after a Greek mythological goddess who rides the rainbows. Many iris flowers exist, e.g., iris unguicularis, iris latifolia and iris tectorum, to name a few. Some of these are easy to distinguish, others are harder. The English statistician and biologist Fisher created a dataset, containing measurable features about three types of iris flowers [44].

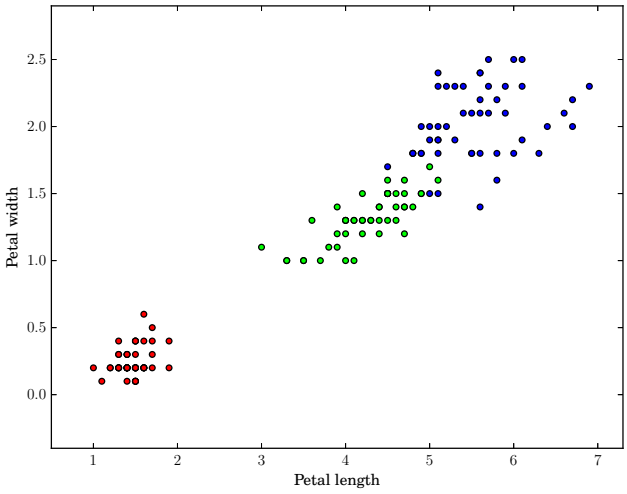


Figure 2.1: Scatter plot of the 'iris' dataset, as provided by [44].

Table 2.1 shows a random sample from the dataset, in the tabular form that data often is represented in. Each row in this dataset represents an iris flower. We call these rows *instances* or *observations*. Because of limited space, we only show a few. The dataset describes for each iris flower some perceptible *features*, such as sepal length, sepal width and petal length (all measured in centimetres). These we call the *attributes* of a dataset. Based on these features, flower experts can determine to which type of iris plants a certain instance belongs (the attribute 'class'). This dataset was created with the purpose of automatically modelling whether an iris flower belongs to the type setosa, versicolor or virginica. That makes this attribute somewhat special, hence we call it the *class* or the *target* attribute.

This is a *numeric* dataset. All attributes (except for the class) contain only numbers, making it easy to plot the data. This is done in Figure 2.1. The x-axis shows the petal length, the y-axis shows the petal width. The attributes sepal length and sepal width are omitted. Each dot represents an iris flower (150 in total), and the colour and shape of a dot shows to which class that flower belongs.

From this plot, we already see that it is quite easy to distinguish the setosa flowers from the others, just by looking at the petal size. We can draw a straight line that separates the setosa flowers from the others. This class is *linearly separable*. However, it is harder to distinguish the versicolor from the virginica. In general, the versicolor flowers have smaller petals than the virginica. But when presented with an uncategor-

ised iris flower with a petal length of 5 cm and a petal width of 1.5 cm, it becomes hard to classify it. One way of solving this problem would be also looking at the sepal length and sepal width, but we can only plot a limited number of variables.

Having only four numeric attributes and a total of 150 instances, from a computational point of view the iris dataset is considered to be an easy dataset to model. Despite the challenges for human experts, machine learning techniques are quite successful at modelling this dataset.

2.2.2 Mushroom

Mushrooms are popular for their edible, medicinal and psychoactive properties. As some mushrooms are very poisonous, knowing which mushrooms are edible and poisonous is quite important. Table 2.2 shows the 'mushroom' dataset. Each row in this dataset represents a mushroom. To keep this table simple, we have left some attributes out. The dataset describes for each mushroom some perceptible features, such as cap colour, odour and gill size. Based on these features, mushroom experts can determine whether a mushroom is edible or poisonous. This dataset was created with the purpose of modelling which properties makes a mushroom edible.

This is a *nominal* dataset. Each attribute can have certain values, for example, the attribute 'stalk surface above ring' can be either silky, smooth or sometimes even fibrous. There is no particular order in the values, making it hard to plot them meaningfully.

Many things can be observed by just looking at this sample. First, we notice some properties about distributions. Although mushrooms come in many colours, it appears that most of them have a red, grey or brown cap. Many mushrooms have either no odour or a foul odour, although some can smell spicy, musty or fishy. The gill colour is more equally distributed amongst the mushrooms, and can be chocolate, pink, buff or something completely different. Moreover, we can already see some correlations between mushroom features and the target. It appears that whenever a mushroom has a foul odour, it is not advisable to eat it. Conversely, from the data sample it seems that when a mushroom has no odour at all, it is edible. However, great care is still advised when eating an unknown mushroom without odour. As this data sample only covers part of the existing mushrooms, in reality many mushrooms have no odour and yet are poisonous. Sometimes, the person collecting the data makes a mistake. A poisonous mushroom can be recorded as edible, or the other way around. Erroneous recorded feature values or class values are called *noise*, which is a common problem data modellers should deal with.

The full dataset contains more than $8{,}000$ instances, and a total of 22 attributes (not including the target). This makes it hard to do a full analysis by hand. In compar-

Table 2.2: Random sample from the 'mushroom' dataset, as provided by [136].

cap colour	odour	gill spacing	gill size	gill colour	stalk surface above ring	stalk surface below ring	stalk colour below ring	spore print colour	population	habitat	...	class
gray	foul	close	broad	chocolate	silky	silky	pink	chocolate	solitary	grasses		poisonous
gray	foul	close	broad	gray	silky	silky	buff	chocolate	several	grasses		poisonous
buff	none	close	broad	white	smooth	smooth	white	white	clustered	waste		edible
gray	none	crowded	broad	pink	smooth	silky	white	white	scattered	grasses		edible
white	none	crowded	broad	gray	smooth	silky	white	white	numerous	grasses		edible
green	none	close	narrow	chocolate	smooth	fibrous	white	chocolate	solitary	woods		edible
gray	foul	close	broad	gray	silky	silky	buff	chocolate	several	paths		poisonous
brown	spicy	close	narrow	buff	silky	silky	pink	white	several	woods		poisonous
brown	fishy	close	narrow	buff	smooth	smooth	white	white	several	woods		poisonous
buff	foul	close	broad	pink	smooth	smooth	white	chocolate	several	grasses		poisonous
white	none	crowded	broad	pink	smooth	silky	white	white	scattered	grasses		edible
red	foul	close	narrow	buff	smooth	silky	white	white	several	paths		poisonous
white	creosote	close	narrow	brown	smooth	smooth	white	brown	several	woods		poisonous
yellow	almond	close	broad	brown	smooth	smooth	white	brown	scattered	grasses		edible
yellow	foul	close	broad	pink	silky	silky	brown	chocolate	several	paths		poisonous
yellow	anise	crowded	narrow	pink	smooth	smooth	white	purple	several	woods		edible
brown	none	crowded	broad	chocolate	fibrous	fibrous	white	brown	abundant	grasses		edible
gray	none	crowded	broad	chocolate	smooth	smooth	white	black	scattered	grasses		edible
red	foul	close	narrow	buff	silky	silky	pink	white	several	woods		poisonous
red	spicy	close	narrow	buff	smooth	smooth	pink	white	several	paths		poisonous
red	none	close	broad	brown	smooth	smooth	gray	black	solitary	woods		edible
red	foul	close	narrow	buff	silky	smooth	white	white	several	woods		poisonous
red	foul	close	narrow	buff	silky	silky	pink	white	several	leaves		poisonous
buff	none	close	broad	red	smooth	smooth	red	white	clustered	waste		edible
gray	none	crowded	broad	black	fibrous	smooth	white	black	abundant	grasses		edible
gray	foul	close	broad	chocolate	smooth	fibrous	white	chocolate	scattered	grasses		poisonous
yellow	anise	close	broad	white	smooth	smooth	white	black	scattered	meadows		edible
white	almond	close	broad	gray	smooth	smooth	white	brown	numerous	grasses		edible
cinnamon	musty	close	broad	yellow	silky	scaly	cinnamon	white	clustered	woods		poisonous
white	pungent	close	narrow	black	smooth	smooth	white	brown	scattered	urban		poisonous
brown	none	close	broad	brown	smooth	smooth	pink	brown	solitary	woods		edible
gray	foul	close	broad	gray	silky	silky	pink	chocolate	several	paths		poisonous
brown	spicy	close	narrow	buff	smooth	silky	white	white	several	woods		poisonous
gray	foul	close	broad	chocolate	silky	silky	buff	chocolate	solitary	woods		poisonous
yellow	foul	close	broad	gray	silky	silky	brown	chocolate	solitary	woods		poisonous

ison to other datasets, such as astronomical telescope data, this is only a very small dataset. We need more sophisticated models and techniques to analyse data sources of increasing size adequately.

2.3 Tasks

As the previous chapter argued, a common machine learning task is to model the available data, such that predictions for new, yet unseen, instances can be made. The task of modelling the 'mushroom' dataset is what we call *binary classification*, as there are only two classes to choose from (the data is *dichotomous*). In the case of the 'iris' dataset, there are already three possible classes. Whenever a dataset has more than two classes, we call the appropriate modelling task *multi-class classification*.

Many real world applications of machine learning are actually multi-class classification tasks, with a rather high number of classes. For example, face recognition programs typically get raw image data as input, and have to determine which person is displayed. In fact, in this case each person in the database is a unique class. Inconveniently, many Machine Learning models are only capable of solving the binary classification task. In order to overcome this limitation, a binary model can be turned into a multi-class model by a divide and conquer strategy called *One-versus-All*. Consider a dataset that consists of n classes, we build $n - 1$ models, such that each separates one class from all the other classes. Allwein et al. [5] propose a framework that consists of various strategies that make binary models useful for multi-class classification. Sometimes, we are not interested in modelling the whole dataset, but in finding an interesting subgroup of the data. The resulting model then describes the data only partially, the other data is out of the scope. This is called *subgroup discovery*.

The previous described datasets contained a nominal target. It is also possible to model datasets with a numeric target. This task type is known as *regression*.

Furthermore, the shown data did not have any concept of time or predefined order. We call it *stationary* data. This is different when modelling, for example, the stock price of a company. Observations obey a certain order, big changes in the economic landscape can have a huge impact on the performance of the model, therefore it needs to be updated constantly. This task type is called *data stream classification* or *data stream regression*. Chapter 5 covers this in detail.

Many other tasks exists, such as *clustering*, *pattern mining* and *association rule discovery*, but as these are out of the scope of this thesis we will not cover them here.

```
1 if (odour = foul) then class=poisonous (2160)
2 if (gill size = narrow) and (gill colour = buff) then class=poisonous (1152)
3 if (gill size = narrow) and (odour = pungent) then class=poisonous (256)
4 if (odour = creosote) then class=poisonous (192)
5 if (spore print colour = green) then class=poisonous (72)
6 if (stalk surface above ring = silky) and (gill spacing = close)
    then class=poisonous (68)
7 if (habitat = leaves) and (cap colour = white) then class=poisonous (8)
8 if (stalk colour above ring = yellow) then class=poisonous (8)
9 otherwise class=edible (4208)
```

Figure 2.2: Decision rule model of the 'mushroom' dataset. Between brackets is the number of instances that are captured by each rule.

2.4 Models

In this chapter, we will describe common Machine Learning methods. Domingos [38] describes a machine learning algorithm as a combination of *representation*, *evaluation* and *optimization*. The representation is the resulting model, the optimization is the procedure that creates that model and the evaluation is the way that model is evaluated (i.e., how good it fits the data). In this chapter we focus on commonly used model types and how they work, rather than on how these are constructed. For each model type, many optimization algorithms exist that are capable of creating them. For simplicity, these algorithms can be seen as black boxes that take data as input and produce a model.

2.4.1 Decision rules

Rule-based models are amongst the most intuitive types of models. Recall that by first inspecting the 'mushroom' dataset, we already came up with some *decision rules*: if the odour is foul then the mushroom is poisonous and if it has no odour then the mushroom is edible.

The rules depicted above are an example of the famous 'One Rule' model [69]. As it uses only one feature, the name is chosen accordingly. It highly simplifies the concept underlying the data, but is already quite accurate. The model gives great insight in what an important property of the problem is. Although the One Rule model adequately captures more than two-thirds of the data sample in this case, typically, a conjunction of many rules is needed to accurately describe the whole concept. Decision rules are commonly used for classification and subgroup discovery.

Some technicalities arise when using decision rule models for classification. For

example, what happens to an instance that is covered by multiple rules, or what happens to an instance that is not covered by any of the rules. Figure 2.2 shows an example of a *decision rule list*. Here, the rules have a certain order. If an instance is captured by multiple rules, the first rule that it complies to is used. Furthermore, it uses the concept of a default rule. All instances that are not captured by any rule are in this case classified as edible.

One of the nice properties about decision rules is that they are typically quite interpretable; human experts can verify and learn from them. Also, in most cases they are quite accurate in modelling the underlying concept.

Many rule induction algorithms exist, e.g., Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [32], Fuzzy Unordered Rule Induction Algorithm (FURIA) [72] and Ripple-Down Rules (RIDOR) [33].

2.4.2 Decision trees

Decision trees are very similar to decision rules. Figure 2.3 shows an example of a decision tree, built upon the mushroom dataset. A decision tree consists of various *nodes*. Tree nodes are drawn round and contain an attribute name; the attribute that is being checked. Leaf nodes are drawn rectangular and contain a class value, in this case whether a mushroom is edible or not. The number between brackets denotes the number of instances that end up in that leaf node. For each observation, we start at the root node and traverse the tree in the direction that the test indicates. For example, for mushrooms that have an almond or anise odour, we traverse the left edge; these appear to be edible. For mushrooms that have no odour, we traverse the middle edge, and end up in the tree node where we will check the spore print colour. This process continues until we end in a leaf node.

When creating a tree, an important question is: which attribute should be used at some point as the splitting criterion. In the seminal paper by Quinlan, the attribute that obtains the highest *information gain* is used [115]. At each node in the tree, we can calculate the entropy $H(X)$ as follows:

$$H(X) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2.1}$$

where p is the number of instances at that tree node belonging to one class, X is the sub-sample of the dataset that is considered in the node (the full dataset at the root, but it is gradually getting smaller at lower levels) and n is the number of instances belonging to the other class. When all the instances belong to the same class, the entropy is 0. When exactly half of the instances belong to both classes, the entropy is 1. In all other cases, entropy is somewhere between the two.

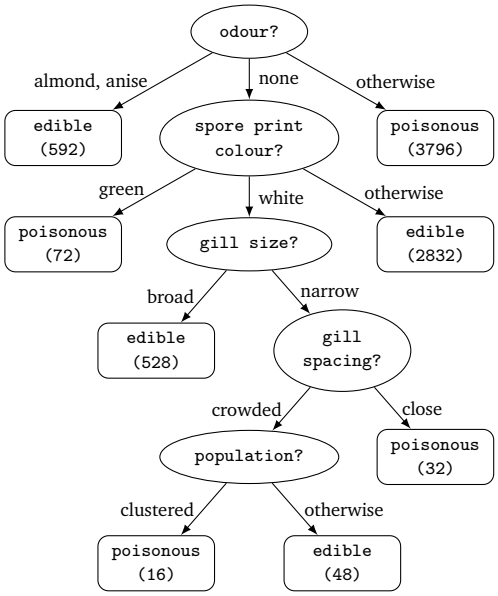


Figure 2.3: Decision tree model of the 'mushroom' dataset.

Suppose we consider an attribute a having v distinct values, and p_i instances of one class and p_n instances of the other class for each $1 \leq i \leq v$. We can determine the entropy after the split in the following way:

$$H'(X) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \left\{ -\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} - \frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i} \right\} \tag{2.2}$$

The attribute that minimizes $H'(X)$ is chosen as splitting criterion. This process is repeated at each node, until a certain stopping criterion is met, e.g., all instances in a node belong to one class.

When using the described procedure, the decision tree is built in a greedy way; sometimes a splitting criterion is chosen that turns out to be suboptimal. As it is proven that building an optimal decision tree is NP-complete [76], we have to resort to such greedy procedures when working with large amounts of data.

As with the decision rule models, decision trees are easy to interpret, and therefore commonly used in practice. Many tree induction algorithms exist, e.g., Classification and Regression Tree (CART) [25], C4.5 [116] and Hoeffding Trees [39].

2.4.3 Probabilistic reasoning

Another way to model the mushroom dataset is by *probabilistic modelling*. In the data sample from Table 2.2, out of the 35 mushrooms, 20 mushrooms are poisonous and 15 mushrooms are edible. We say that the *prior probability* of a mushroom being poisonous $P(\text{class} = \text{poisonous}) = 20/35 \approx 0.57$ and the prior probability of a mushroom being edible $P(\text{class} = \text{edible}) = 15/35 \approx 0.43$. This observation gives us reason to believe that in general a mushroom is more likely to be poisonous than edible. However, for each individual mushroom, we can adapt this probability based on the observed features. For example, we observe that the mushroom has a smooth stalk surface above the ring. We immediately observe that the prior probability of a mushroom having a smooth stalk surface (above the ring) $P(\text{stalk surface} = \text{smooth}) = 21/35 = 0.6$. We want to know the probability of a mushroom being poisonous, given the fact that is has a smooth stalk surface. Conveniently, Bayes' theorem states that given a hypothesis and evidence that bears on that hypothesis:

$$P(\text{hypothesis}|\text{evidence}) = \frac{P(\text{evidence}|\text{hypothesis}) \cdot P(\text{hypothesis})}{P(\text{evidence})} \tag{2.3}$$

In this case, the evidence is a smooth stalk surface and the hypothesis is that the mushroom is poisonous. In fact, the hypothesis could as well be that the mushroom is edible, as this is the exact opposite of the previous hypothesis.

Of all the 20 observed poisonous mushrooms, 8 had a smooth stalk surface above the ring. The *likelihood* of a smooth stalk surface above the ring given that the mushroom is poisonous $P(\text{stalk surface} = \text{smooth}|\text{class} = \text{poisonous}) = 8/20 = 0.4$. Likewise, of the 15 edible mushrooms, 13 had a smooth stalk surface above the ring. Thus, the likelihood of a smooth stalk surface above the ring given that the mushroom is edible $P(\text{stalk surface} = \text{smooth}|\text{class} = \text{edible}) = 13/15 \approx 0.87$.

Plugging these numbers in Eq. 2.3 results in a probability of a mushroom being poisonous given that the stalk surface (above the ring) is smooth:

$$P(\text{class} = \text{poisonous}|\text{stalk surface} = \text{smooth}) \approx \frac{0.4 \cdot 0.57}{0.6} \approx 0.38 \tag{2.4}$$

Likewise, the probability of a mushroom being edible given that the stalk surface is smooth:

$$P(\text{class} = \text{edible}|\text{stalk surface} = \text{smooth}) \approx \frac{0.87 \cdot 0.43}{0.6} \approx 0.62 \tag{2.5}$$

This model scales trivially to multiple attributes; when classifying a mushroom, we should not only take into account the stalk surface (as we did in the previous example), but all other attributes that seem to be of influence. In the case of the mushrooms, we should also use odour, gill size, and probably even more. With probabilistic

modelling, the main challenge lies in identifying which attributes are important for classifying an instance.

There are many algorithms capable of building a probabilistic model. One of the most well-known is Naive Bayes, which is built upon the 'naive' assumption that all attributes are independent from each other, i.e., they do not interact. More sophisticated models can be built by means of Bayesian Networks. Creating an optimal Bayesian model is NP-complete [31, 34]. This implies that we have to rely on greedy or heuristic techniques when modelling large datasets.

2.4.4 Nearest Neighbour models

The main idea of *Nearest Neighbour models* is to identify and store some *key* instances from the dataset. Whenever presented with a new instance, find among the remembered instances the ones that are most similar to this new instance [35].

Two important issues arise. First, how do we define which are the key instances, and second, how do we determine which of the instances are most similar.

In the case of the 'iris' and 'mushroom' dataset, the issue of the key instances can be easily resolved. As both datasets contains only a small number of instances, all can be maintained in memory. However, with bigger datasets this becomes a serious issue that needs to be addressed, for example by sampling random instances.

One of the most commonly used measures of similarity is the Euclidean distance. Formally:

$$dist(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_k - x'_k)^2} \tag{2.6}$$

Here, x is the new instance, x' is one of the key instances, and x_i and x'_i denote the value of a given attribute, with $i = (1, 2, \ldots, k)$.

Having a similarity measure, it is easy to find the instances that have the lowest distance to x. Many other distance functions can be used as well. This technique is called *k-Nearest Neighbours*.

When creating a nearest neighbour model such as the one described, the data needs to be prepared with care. When dealing with attributes that are on different scales, the attribute with the biggest scale often dominates the Euclidean distance. Therefore, the data is often normalized to get the values of all attributes within the same interval. Moreover, all distance-based models suffer from a concept that is called the *curse of dimensionality* [50]. As high-dimensional datasets tend to be extremely sparse, the data points are often far away from each other. As such, instance-based models are also vulnerable to irrelevant attributes. It is recommended to use these models in combination with a feature selection technique [111].

2.4.5 Logistic Regression

Regression is a commonly used technique to model the relationship between numeric input variables and a numeric target. However, it can also be used for classification. In that case, *Logistic Regression* is used. Logistic Regression is a technique that models the probability of a newly observed instance belonging to a certain class. One interesting property is that it gives a degree of certainty that an instance belongs to a certain class. For example, consider the 'iris' dataset (Table 2.1). When classifying the third example, we would be much less certain that it actually belongs to the class virginica than when classifying, e.g., the fourth example. This is because the third example is much closer to the so-called *decision boundary*.

A typical regression model usually has the form:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_k x_k \tag{2.7}$$

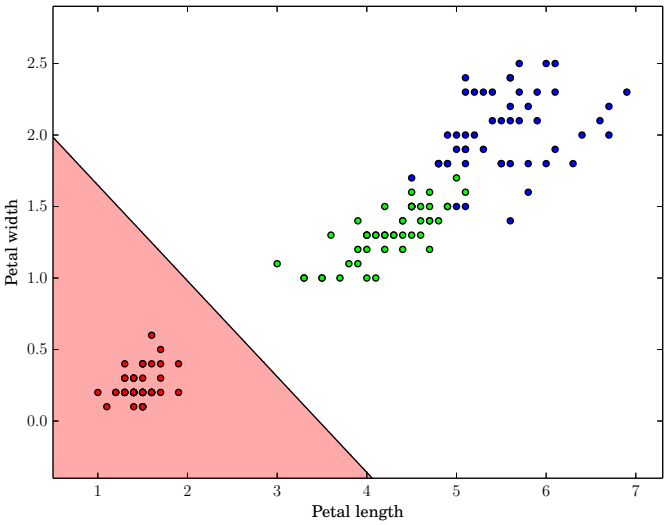
where x_i are the attribute values, w is a weight vector and y is the target.

By setting $y = 0$, we get a line or (hyper)plane that represents the model. In the case of normal regression, this line or (hyper)plane aims to fit the data points. In the case of Logistic Regression, this line or (hyper)plane separates the various classes, and is therefore called the *linear discriminant*. Figure 2.4 shows a Logistic Regression model built upon the petal width and petal length attributes of the 'iris' dataset.

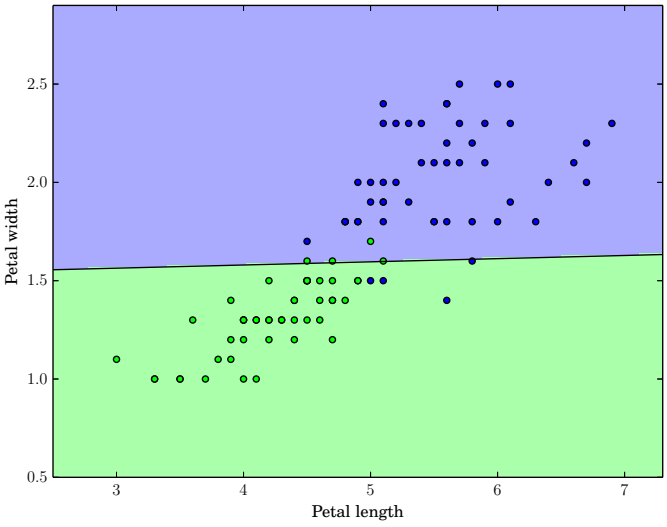
As the 'iris' dataset is a multi-class classification task, we can not separate them using one linear discriminant. Instead, we use the One-versus-All strategy to first separate instances of the class setosa from the others ($y = 3 + -1 \cdot \text{length} + -1 \cdot \text{width}$, see Figure 2.4a). If an instance does not belong to the setosa class, we can further establish whether it belongs to the versicolor class ($y = 4.3 + -0.55 \cdot \text{length} + -1 \cdot \text{width}$, see Figure 2.4b). If it does not belong to that class either, it will be classified as virginica.

The values of any new instance can be plugged into the respective formulas, resulting in a value ranging from $[-\infty, \infty]$. If the outcome is bigger than 0, it means it belongs to the specified class. If it is smaller than 0, it belongs to the other class (or set of classes). When the value is exactly 0, the model does not know to which class it belongs, and will have to guess. Conveniently, we can use the *logistic function* to map these back to the interval $[0, 1]$, in order to obtain proper probability estimates.

Logistic Regression requires a dataset to be linearly separable to perfectly fit the training data. For the 'iris' dataset this is not the case, as can be seen from Figure 2.4b. The resulting model therefore classifies some of the instances wrongly.



(a) Setosa vs. the rest



(b) Versicolor vs. Virginica

Figure 2.4: Logistic Regression model of the 'iris' dataset.

2.4.6 Support Vector Machines

Similar to Logistic Regression models, a *Support Vector Machine* (SVM) is a (hyper)plane-based model that separates the classes. Typically, there are many lines or (hyper)planes that do so. Support Vector Machines maximize the margin around the linear discriminant. The data points that lie closest to the decision surface are typically the most difficult to classify. These are called the *support vectors*. Based on these, the separating hyperplane is calculated. The distance between the support vectors and the separating hyperplane is maximized. This distance is called the *margin*.

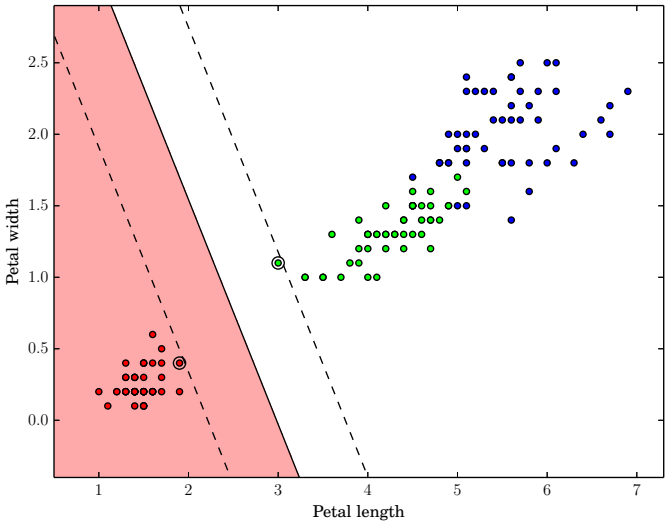
Figure 2.5 shows an example of a Support Vector Machine built upon the 'iris' dataset. Support Vector Machines are suitable for binary classification tasks. Therefore, in this case it uses the One-versus-All strategy to first separate the setosa class from the others (Figure 2.5a), after which it separates the Versicolor from the Virginica (Figure 2.5b). As with Logistic Regression, it requires linear separability. Otherwise, misclassifications already occur in the training set. In Figure 2.5b we see 5 misclassified instances, and even 8 instances that fall within the separation margin.

Often it occurs that the data is not linearly separable in the original representation, but is linearly separable when you represent it differently. Support Vectors Machines are often used in combination with a *kernel*. Kernels map the data in a higher dimension, effectively resulting in more attributes. For the task of finding the separating hyperplane and classifying new instances, we only need the dot product of the original features, saving us from additional memory usage. Popular kernels for classification purposes are the *Radial Basis Function kernel* (RBF), *Polynomial kernel* and *Sigmoid kernel*. Mapping the data to a higher input space should be done with great care, due to the curse of dimensionality. Increasing the number of variables, exponentially increases the number of possible solutions, yielding many sub-optimal solutions [98].

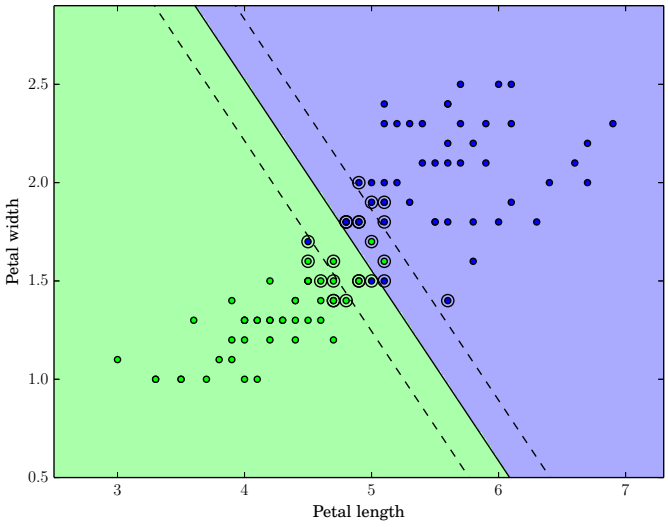
A Support Vector Machine model is based on a small amount of data points, making it a rather stable model. Adding or removing data points does not affect the model, unless that data point is in fact one of the support vectors. Support Vector Machines are typically not affected by local minima. The name of Support Vector Machines can be misleading. It is not a machine, it is a model. When using the right kernel, Support Vector Machines are very powerful models, building upon a solid theoretical foundation.

2.4.7 Neural Networks

An alternative modelling approach is based on the human brain. The human brain is a collection of billions of *neurons*, that are ordered in a certain graph structure. Each neuron has certain inputs and outputs. A neuron outputs a signal if the combined

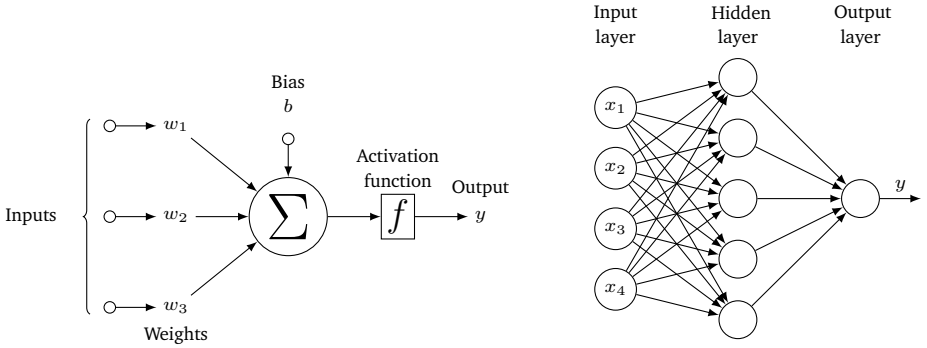


(a) Setosa vs. the rest



(b) Versicolor vs. Virginica

Figure 2.5: Support Vector Machine model built upon the 'iris' dataset. The solid line is the separating hyperplane, the circled data points are the support vectors. The striped line indicates the margin of the separated hyperplane.



(a) A single neuron

(b) Structure of a network with 1 hidden layer that contains 5 neurons

Figure 2.6: Example of a neural network.

input signals exceed a certain threshold. Inspired by the success of the human brain, machine learners imitated this paradigm by means of *Neural Networks*.

Figure 2.6 shows an example of a neural network. Figure 2.6a shows an abstraction of a neuron. This neuron has 3 preceding input neurons, which are all associated with a certain weight. Each neuron has a certain threshold value, here displayed with b. The output value y of each neuron is determined by adding all the weighted inputs together; the result is put in the *activation function*. The activation function f has two purposes. First, it determines whether the threshold value is exceeded or not, and second, it maps the resulting output of the neuron in the desired domain, typically $[0, 1]$.

This determines the outcome of the neuron. Typically, neural networks are built as acyclic graphs, consisting of distinguishable layers. These networks are called *feed forward networks*. Figure 2.6b shows an example of the network structure. It contains an input layer, one hidden layer and an output layer. Typically, the feature values of a given instance are fed into the input layer; all the other layers are the neurons as described before.

Indeed, a network that contains no hidden layers (and thus only has one real neuron, i.e., the output node) has a model corresponding to Linear Regression: each input variable is associated with a certain weight, and the bias of the neuron corresponds to w_0 of the Logistic Regression model. These neural networks are called *perceptrons*.

The real power of neural networks lies in the hidden layers. Effectively, these are capable of simulating derivatives of feature combinations, that are not manifested in

the original input space, giving it a tremendous expressive power. Great care should be taken with noisy data. Due to the high expressive power, it can easily learn a wrong concept from the noise. The biggest challenges in neural networks is finding a good network structure and the right weights for each neuron. Various algorithms are proposed for this, most commonly back-propagation. Training a 3-node neural network is already NP-complete [17].

2.5 Evaluation

For a created model, an important issue that needs to be addressed is, how well does it perform. In order to do so, we need an *evaluation measure*, which quantitatively defines performance. We could give the model some instances of which we already know the correct class label, and see in what percentage of cases it classifies them correctly. These instances are called the *test set*. This measure is often called *predictive accuracy*, and one of the most natural ways of measuring the performance. Predictive accuracy is a measure that is calculated over the whole test set. It is important to measure the *generalization* capabilities of a classifier, i.e., how good the predictions are for instances that it has not been trained on.

Sometimes, there is an asymmetry in the importance of misclassifications. For example, it is worse to classify a poisonous mushroom as edible than the other way around. Likewise, banks determine whether a person is eligible for receiving credit based on social-economic attributes such as employment status, age and credit history. From their perspective, it is worse to give credit to someone that is not creditworthy, than the other way around. The 'German credit' dataset contains past records of credit applications; the goal is to model when someone is eligible for receiving credit. For this purpose, we could use class-specific evaluation measures, that only address the performance on a given class. The *sensitivity* or *true positive rate* (TPR) is the proportion of instances that belong to a certain class that are correctly classified as such. The *specificity* or *true negative rate* (TNR) is the proportion of instances that do belong to a certain class and that are correctly classified as such. Typically, there is a trade-off between the sensitivity and specificity. For example, when a classifier always predicts a given class, it will have a perfect true positive rate, but a true negative rate of 0, and vice versa. A similar trade-off between the true positive rate and the false positive rate is depicted in the *receiver operating characteristic curve* (ROC-curve), a common graphical evaluation measure.

Figure 2.7 shows the ROC curve of three classifiers that model whether someone is credible or not. Banking companies that want to model future credit applications can choose between Naive Bayes and the Nearest Neighbour approach. Although Naive

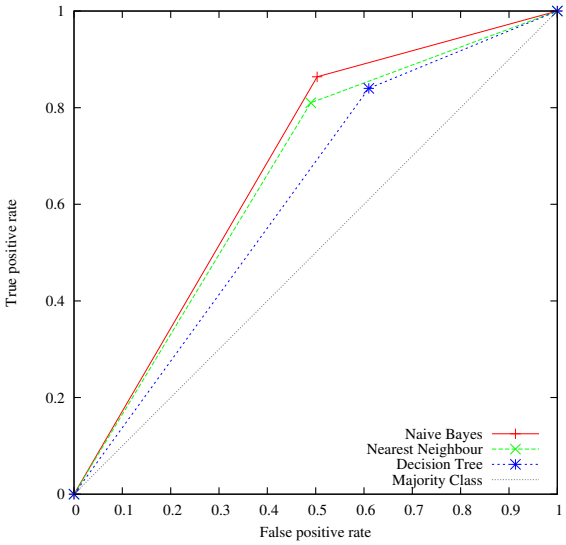


Figure 2.7: ROC curves of three classifiers on the 'German credit' dataset. The grey line shows a virtual classifier that always predicts the same class.

Bayes has a higher accuracy (this can not be deduced from the image), the nearest neighbour method might be selected based on the ROC curve; not credit-worthy persons will be classified as credit-worthy less often. The decision tree model is *dominated* by the Naive Bayes model, it obtains a lower true positive rate and a higher false positive rate.

It is important to always evaluate the performance of a model based on data that the model is not trained on. It is easy to see that by memorization it is easy to perform well on the full training set. What we are interested in is generalization beyond this. For this reason, typically a *holdout set* is used: the model is trained based on a percentage of the data, and then evaluated on the other part of the data.

There are two drawbacks to this approach. First, the model is not tested on all instances. When the instances that are hard to classify all end up in either the training set or the test set, this will give respectively an optimistic or pessimistic assessment of the model. Second, as the model is only evaluated once, it does not give a stable assessment of performance.

In order to overcome these problems, n-fold *cross-validation* is usually preferred. The data set is split in split in n equal subsets, and n different models are trained using $n-1$ of these subsets and tested on the remaining one. This way, it tests the model using each instance exactly once. Cross-validation is widely considered a reliable way

of evaluating a model on a single dataset.

However, from a Machine Learning point of view, it is often desirable to make more general conclusions, based on evaluations over many datasets. To this end, statistical tests are used. This allows making statements about the performance of classifiers over multiple datasets or cross-validation runs. Demšar [36] reviews various statistical tests appropriate for assessing the quality of Machine Learning models and algorithms.

2.6 Discussion

This chapter showed some examples of common Machine Learning data, tasks and models. We have seen various models, all leading to a different representation and therefore a different expressive power and bias towards certain kinds of data. Having more expressive power does not guarantee better classification performance. Sometimes highly complex models over-fit on an irrelevant concept and perform unexpectedly mediocre.

One aspect that is nearly not covered are algorithms. These are sets of rules to be followed in order to create the models. For each model type, there are many algorithms that are able to create such models. Each of these algorithms in turn contains various parameters to be determined by the user; these parameters enable small or big nuances in the resulting model. This makes it hard for anyone interested in modelling a dataset to select the appropriate model, algorithm and parameter settings. One question that arises from this is, can we learn from previous machine learning experiments what kind of algorithm should be used to model a new dataset; this is called *meta-learning*. As machine learning aims to model a certain dataset, meta-learning aims to model what kind of data should be modelled by what kind of technique. In the next chapter, we will review common techniques of doing so.

3 Meta-Learning

Machine Learners have been very successful at integrating data-driven methods in many other domains and sciences: Chemist model the activity of chemical compounds by means of Machine Learning [81], popular media companies use knowledge obtained from previous data to recommend new content to their subscribers, and modern cars use Machine Learning techniques to take away the burden of parking.

However, when it comes to Machine Learning itself, decisions are seldom made by what previous gathered data has learned. When presented with a new problem, often a solution is chosen by trial and error. Typically, a small set of algorithms is selected by intuition, and the best of these is selected to solve the problem. This trial and error is sometimes called 'the black art of Machine Learning' [38], implying that a more scientific approach is desirable.

A huge challenge lies in solving Machine Learning problems in a data-driven way. This scientific challenge is called *meta-learning*; the research field that aims to learn from previous applications and experiments.

3.1 Introduction

The field of meta-learning has attracted quite some attention, and quite some problems and challenges in various directions have been addressed. Although it is impossible to capture the whole field in one single definition, we will consider the definition by Vilalta et al. [159]: "The field of meta-learning has as one of its primary goals the understanding of the interaction between the mechanism of learning and the concrete contexts in which that mechanism is applicable." In this definition, the 'mechanisms of learning' are the algorithms (that build models) and the 'contexts in

which that mechanism is applicable' are the Machine Learning tasks and datasets. Basically, we want to obtain knowledge (or learn) which algorithm and parameter setting should be used on what kind of data.

This problem can be viewed from many different perspectives. Most notable, the *algorithm selection problem*, is defined as: given a dataset, which algorithm and parameters will obtain maximal performance according to a specified measure [119]. As the amount of algorithm and parameter combinations is infinite, this in itself is already a hard but important problem. Considering common applications of Machine Learning, e.g., in healthcare and epidemiology, performing slightly better on a given task can already result in making the difference for human lives.

In many realistic settings, the algorithm selection problem can also be seen as a *search problem*. A model, algorithm and parameter settings are recommended, and these are being tested (either in production or using cross-validation). For example, a company modelling customer behaviour, could already apply the recommended model in their production environment. However, they can still continuously test other models to find one that improves upon the original selected model. This process is repeated until a satisfactory model has been found.

Sometimes, the model is fixed a priori because of, e.g., empirical performance evidence, or interpretability requirements. In this case, the main challenge is to select the appropriate hyperparameters. Hence, this task is called *hyperparameter optimization* (e.g., [8, 75, 89]). Many techniques from the optimization literature can be used for this, e.g., Particle Swarm Optimization, Evolutionary Algorithms or Bayesian optimization.

Rather than looking at algorithm performance, we can also look at dataset properties. Where meta-learning is often focussed on categorizing datasets as a whole entity, Smith et al. [140] focusses on individual instances, attempting to categorize which are often misclassified, why this happened and how they contribute to data set complexity. This research potentially improves the learning process and can guide the development of learning algorithms.

The remainder of this chapter is organised as follows. We discuss three general approaches to algorithm selection. Section 3.2 approaches this from the learning paradigm; Section 3.3 approaches this from the search paradigm; Section 3.4 introduces the notion of ensembles. Next, we introduce some important aspects to meta-learning. Section 3.5 discusses the 'Law of Conservation for Generalization Performance', which is sometimes erroneously cited to dismiss meta-analysis. Section 3.6 discusses some analytically obtained knowledge about models, which is a very general form of meta-knowledge. Section 3.7 examines the bias variance trade-off, and the dilemma that comes with choosing between over-fitting and under-fitting. Section 3.8 concludes with a discussion.

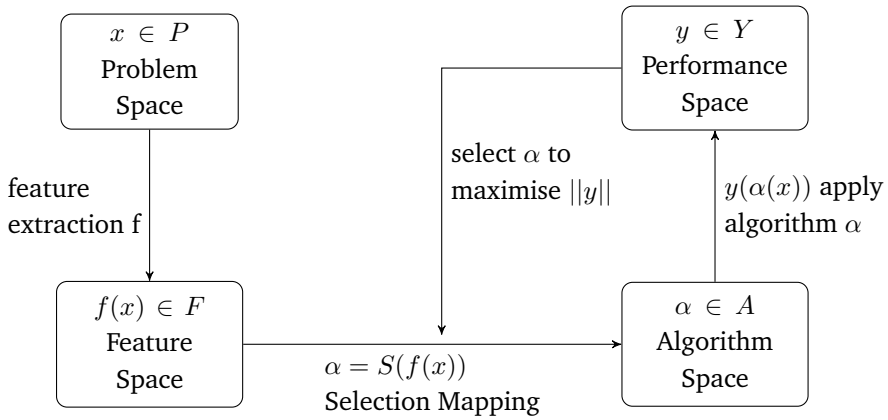


Figure 3.1: The Algorithm Selection Framework, taken from [141].

3.2 Learning Approach

The algorithm selection problem occurs in many other fields besides Machine Learning. Many optimization problems can be solved with a wide range of algorithms and parameters, and therefore there is room for algorithm selection. For example, satisfiability problems, travelling salesman problems and vehicle routing problems.

The algorithm selection framework, formalised by Rice [119], addresses this. The framework is illustrated in Figure 3.1. According to this definition, the problem space P consists of all problems (tasks) from a certain domain. Each problem $x \in P$ can be mapped by means of some feature extraction procedure $f(x)$ into the feature space F. The feature space F contains measurable characteristics calculated upon these problems, e.g., the number of instances or the number of attributes. We call these *meta-features*. The algorithm space A is the set of all considered algorithms that can be used to solve these tasks. And finally, the performance space Y represents the mapping of these algorithms to a set of performance measures. The task is for any given $x \in P$, to select the algorithm $\alpha \in A$ that maximizes a predefined performance measure $y \in Y$.

Essentially, this itself is a learning problem, and can be solved using conventional Machine Learning techniques. Often, Random Forests are used [144]. Like with any learning problem, there are instances, represented by features that have a certain class feature. In this case, the best performing algorithm is the class attribute. This task can also be casted as a ranking or a regression problem. In the case of the ranking problem, the goal is to order the classifiers by their expected performance; in the case of the regression problem, the goal is to predict the expected performance for a set of

given algorithms.

Indeed, by approaching the algorithm selection problem as a learning problem, all Machine Learning algorithms can be used as meta-learner and solve the algorithm selection problem. Various solutions solving this problem in novel ways have been proposed (see, e.g., [4, 27, 87, 144]).

Some non-trivial considerations remain. Most importantly, as with any modelling task, the set of (meta-)features determines the quality of the solution. These need to be chosen and constructed appropriately. Furthermore, the performance space determines on what performance criteria the algorithm should be selected.

3.2.1 Feature space

The performance of a meta-learning solution typically depends on the quality of the meta-features. Typical meta-features are often categorized as either simple, statistical, information theoretic, algorithm/model-based or landmarkers.

The simple meta-features can all be calculated by one single pass over all instances and describe the data set in an aggregated manner, e.g., number of instances, number of attributes and number of classes [20]. The statistical meta-features are calculated by considering a statistical concept (e.g., standard deviation, skewness or kurtosis), calculate this for all numeric attributes and taking the mean of this. This leads to, e.g., the mean standard deviation of numeric attributes. Other statistical aggregation methods can also be used instead of the mean, e.g., the minimum, the maximum, or the median. Likewise, the information theoretic meta-features are calculated by considering an information theoretic concept (e.g., mutual information or attribute entropy), calculate this for all nominal attributes and taking the mean of this. This leads to, e.g., mean mutual information. Again, other statistical aggregation methods can be used as well. By aggregating the statistical and information theoretic concepts, information is lost by definition. A main challenge lies in finding a way of preserving this information. Sometimes the meta-data is built upon datasets with the exact same features. In that case, no aggregation over the meta-features is needed; these can be calculated and used for every individual features, without losing any information [164].

Landmarkers are performance evaluations of fast algorithms on a dataset [108]. Sometimes, knowledge about how simple classifiers perform on a dataset yields information on the performance of the more complex and time-consuming algorithms. However, these procedures should be used with great care, as the time for calculating the meta-features should not exceed the time of bluntly running all algorithms on the dataset.

Taking this idea one step further leads to model-based features [104]. Again, a

simple model is built upon the data, most commonly, decision trees. Topological properties from this model can be used as meta-features, e.g., shape, number of leafs, or maximum tree depth.

Sun and Pfahringer [144] propose pairwise meta-rules. These are simple decision rules that determine for each pair of algorithms, which will work best under which conditions. This type of meta-feature assumes that plain information from landmarkers is not necessarily represented best in its numeric form, and transforms this information to a binary attribute, at the cost of additional computation time.

It has proven hard to come up with an appropriate set of meta-features. Therefore, Provost et al. [113] proposes Partial Learning Curves as an intuitive way of mapping a problem into the feature space. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size. Intuitively, we have now information on how the actual classifiers that we are interested in work on the actual dataset that we are interested in. Various methods have been proposed that exploit partial learning curves to solve the algorithm selection problem [86, 87, 127].

Pinto et al. [110] propose a framework that enables the systematic generation of meta-features specific to a certain domain. The framework detects what kind of meta-features can lead to the generation of new meta-features. For example, when using a decision tree landmarker, this can trigger the generation of model-based meta-features extracted from the decision tree. Their results indicate that sets of systematically generated meta-features are more informative and contain more predictive power than ad-hoc selected meta-features.

3.2.2 Performance space

Commonly, solutions to the algorithm selection problem focus on finding an algorithm that maximizes predictive accuracy. This makes sense for a variety of theoretical and pragmatic reasons; it objectively orders the full set of available algorithms, is easy to establish (e.g., by means of cross-validation) and is what we are commonly interested in [59]. However, as there are many tasks where almost all algorithms perform well in terms of accuracy [69], it makes sense to also consider other ways of selection criteria for algorithms. Some options for this are computational complexity, compactness of the resulting model or comprehensibility of the resulting model [59]. Sometimes it is sensible to make a trade-off between accuracy and run-time [21, 127]. In some problem domains, the balance between the classes is skewed. Although in those cases it might be easy to obtain a high accuracy by predicting the majority class, it is more interesting to have a model that performs well on the other class(es). Measures such as area under the ROC curve, precision and f-measure are decent options for those problems.

Furthermore, it is debatable whether the single best model should be recommended. Alternatively, a statistical test could be performed, and all algorithms that perform statistically equivalent to the classifier obtaining the highest score are considered good recommendations. Intuitively, this makes sense: if there is no statistical evidence that there is a difference between the performance of the algorithms, then there is no strong evidence to prefer one over the other. In many modern applications, it is not enough to predict a single algorithm. Rather than predicting one single algorithm, a ranking is produced, giving alternatives if the first advised algorithm does not perform adequately. The algorithm selection problem could even be seen as a regression problem: for each algorithm $\alpha \in A$ an estimated performance should be predicted, and the algorithm with the highest estimated score can be selected.

3.3 Search Approach

Alternatively, the algorithm selection problem can be cast as a search problem, which can be solved by many techniques from the field of black-box optimization. Typically, an optimization algorithm recommends an algorithm and parameter setting, and it is then tested using some evaluation procedure, e.g., cross-validation. It continues on recommending these procedures, until the budget runs out. Then the best performing algorithm and parameter setting combination is selected.

In some cases the task is to find the best algorithm and parameter setting, in other cases the algorithm is fixed a priori, and the task is to find the optimal parameter settings. This task is called *hyperparameter optimization*.

A commonly used search strategy for hyperparameter optimization is *grid search*, which bluntly tries all possible parameter settings. As many parameters accept continues values, for most algorithms there are infinitely many parameter settings. Therefore grid search requires the intervention of a human expert, who selects sensible parameters, ranges and discretizations. Grid search has no natural way of dealing with a budget.

To overcome these limitations, *random search* could be used as an alternative. As the name suggests, it randomly tries some parameter settings from all possibilities. This way, there is no real need for the human expert to select parameters, ranges and discretizations (although it can still benefit from the input of a human expert). Furthermore, it naturally deals with a budget, as it can just stop whenever the budget runs out. The best parameter settings so far will be selected. Random search often performs better than grid search. When exploring the same parameter space it finds an acceptable setting much faster, and when using a fixed budget it goes beyond the selected parameters and ranges that grid search is restricted to [8].

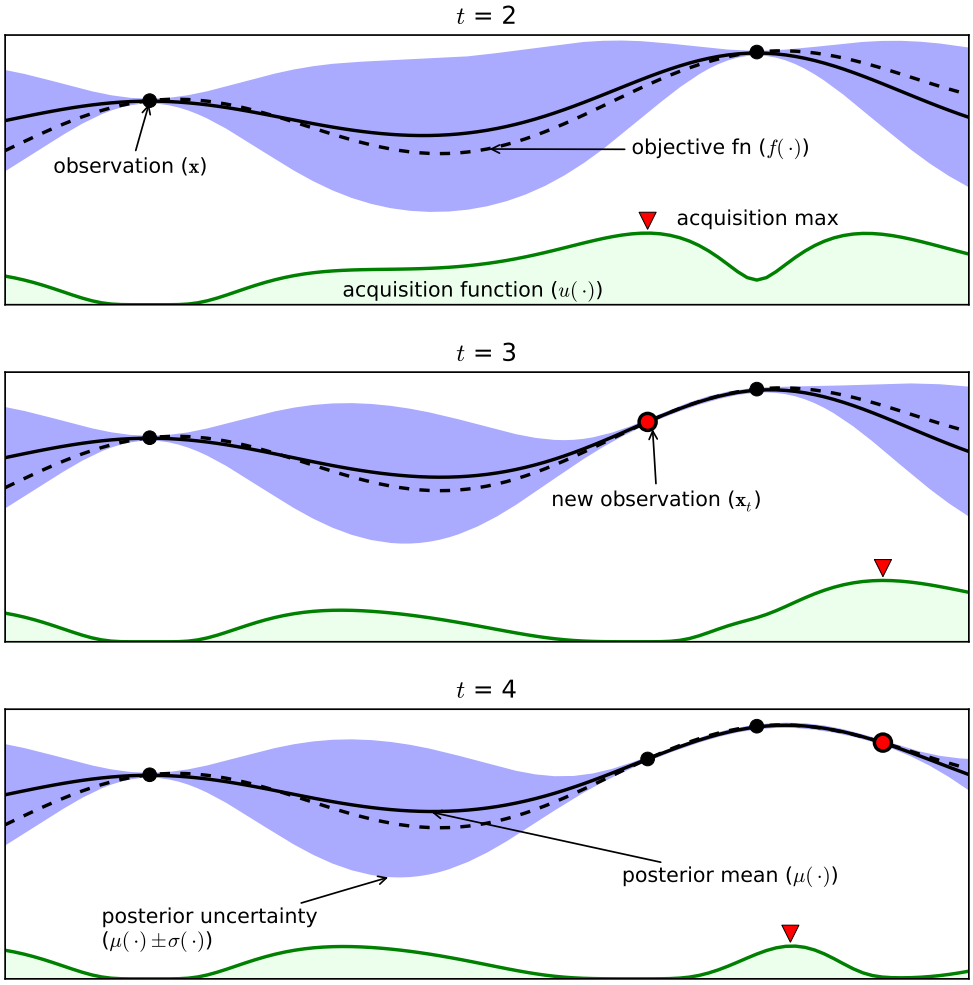


Figure 3.2: Example of Bayesian Optimization on one variable. The dashed line represents the objective function, which we only know partly (at the black dots); the striped line represents the posterior mean, and the purple area the posterior uncertainty; the acquisition function is plotted in green. Image taken from [26].

Both grid search and random search are *static search methods*, information obtained from results of earlier test is ignored. Opposite to this, *dynamic search methods* attempt to use this information. For example, when a certain parameter setting obtains good results, it is plausible that similar parameter settings will also obtain good results and possibly improve on the current best result.

Bayesian Optimization is a strategy that attempts to exploit knowledge obtained from earlier tests [26]. The parameter space is often modelled as a *Gaussian process*; it assumes that the underlying function is smooth, i.e., small fluctuations in the parameter settings will lead to only small fluctuations in the result. A so-called *acquisition function* determines which parameter setting is evaluated next based on high uncertainty and high potential. Figure 3.2 illustrates this in an example, in which one parameter is optimized. At each point t, it selects a new parameter setting that is being evaluated. Sequential Model-based Bayesian Optimization extends this idea, also selecting among various learning algorithms [75]. Notably, Auto-Weka applies this on Machine Learning, automatically searching for a good algorithm and parameter setting among all algorithms from the toolbox WEKA [147].

The search problem can also be modelled as a *Multi-armed Bandit problem* [68, 89], named appropriately after a gambler who's aim is to select a slot machine that gives him maximal reward. Typically there is a set of *arms*, each representing an algorithm with parameter configuration. Pulling an arm is associated with a certain cost (training a model on an amount of data) and reward (the performance of this model). Pulling a certain arm multiple times corresponds to training that model on an increasing amount of data, gaining a higher confidence in the measured performance of this model. Two notable algorithms are Successive Halving and Hyperband [89]. These start with testing a large number of algorithms on a small amount of data; badly performing algorithms are eliminated and good performing algorithms are tested with more data. This works very well in practise. Additionally, these algorithms come with theoretical guarantees about the maximum *regret*, i.e., the difference between the recommended algorithm and parameter setting and the absolute best one.

Despite these sophisticated techniques and theoretical guarantees, the simplicity of Random Search still proves to be a strong baseline. Given twice the budget, it often outperforms guided search schema's [9].

3.3.1 Combining Search and Learning

Various strategies have been proposed that combine the thoroughness of the search strategies with the knowledge-based approach of meta-learning.

Active Testing combines grid search with meta-knowledge [2, 88], intelligently selecting the order in which the algorithm and parameter combinations are tried. It aims to minimize the number of algorithms that need to be evaluated before an adequate model has been built. It assumes that there is a meta-dataset containing information on how the algorithms and parameter settings under consideration performed on other problems. Furthermore, it assumes that already one algorithm is selected as the most promising to try first. This could be the one that performed best on historic

data, an algorithm recommended by another meta-learning system, or even one that was randomly selected. This algorithm is called the *current best*. From then on, the algorithm that outperforms the current best algorithm on most historic datasets that seem similar to the current dataset, is tested next. For each new algorithm to try, it only focusses on historic datasets on which the new algorithm outperforms the current best. This is inspired by the idea that we are interested in volatile algorithms. Naturally, if the newly tried algorithm turns out to be better than the current best, from that moment on it is the current best. This process is repeated until an appropriate algorithm has been selected or a predefined budget (e.g., run time) runs out. Chapter 6 details on this method.

Alternatively, meta-learning can be used to initialize complex search methods. It has been known that Sequential Model-based Bayesian Optimization converges fast when its initial points yield already good performance. Hence, meta-learning can be natively used to find promising parameter settings on similar datasets [43]. These can be used as initial evaluations for the Sequential Model-based Bayesian Optimization, leading to faster convergence.

3.4 Ensembles

Another approach to select the best model, is to combine multiple models in an *ensemble* of classifiers. Ensemble techniques train multiple classifiers (also called *members*) on a set of weighted training examples; these weights can vary for different classifiers. In order to classify test examples, all individual models produce a prediction, also called a *vote*, and the final prediction is made according to a predefined voting policy. For example, the class with the most votes is selected. Dietterich [37] identifies three reasons why ensembles work better than individual models.

- **Statistical**. When there is insufficient training data, there will be multiple models that fit the training set well, but have various (unknown) performance on the test set. Combining multiple models spreads the risk of a misclassification among multiple models.

- **Computational**. Even when there is sufficient data, the learning algorithm that induces the model might get stuck in a local optimum. Running the learning algorithm multiple times from various starting points may give a better performance on the unknown test set.

- **Representational**. In many Machine Learning applications, the true concept that is being modelled can not be represented by a given algorithm. For example, Logistic Regression can not represent the XOR-function, because it is

not linearly separable (see Chapter 2.4.5). However, an ensemble of multiple Logistic Regression models is able to perfectly represent it.

Condorcet's jury theorem [83] gives theoretical evidence that the error rate of an ensemble in the limit goes to the Bayesian optimum (i.e., the maximum obtainable accuracy) if two conditions are met. First, the individual models must do better than random guessing. Second, the individual models must be diverse, i.e., their errors should not be correlated. If these conditions are met, increasing the ensemble size decreases the amount of misclassifications [64]. Indeed, if the models do worse than random guessing, increasing the ensemble size also increases the amount of misclassifications.

Basically, two approaches of model combination exist [22]. The first one exploits variability in the data, and trains similar models on different subsamples of the data. The second one exploits variability among models, and trains fundamentally different models on the same data. *Bagging* [23] exploits the instability of classifiers by training them on different subsamples called *bootstrap replicates*. A bootstrap replicate is a set of examples randomly drawn with replacement, to match the size of the original training set. Some examples will occur multiple times, some will not occur in the bootstrap replicate. Bagging works particularly good with *unstable* algorithms, were small fluctuations in the training set lead to dissimilar models. Bagging reduces the variance error of a model, and only slightly affects the bias error. Algorithms that have a high variance error typically perform much better when used in a Bagging setting, at the cost of losing interpretability.

Boosting [135] is a technique that corrects the bias of weak learners. A *strong learner* is one that produces a highly accurate model; a *weak learner* is one whose models perform slightly better than random guessing [149]. In his seminal paper, Schapire [135] shows weak learners and strong learners are equivalent; he presents an algorithm that combines various weak learners that combined perform as a strong learner. Boosting sequentially trains multiple classifiers, in which more weight is given to examples that where misclassified by earlier classifiers. It decreases both bias and variance error. Some common forms of boosting are Adaptive Boosting [45], Logistic Boosting [48] and Gradient Boosting [51].

Stacking [163] combines heterogeneous models in the classical batch setting. It trains multiple models on the training data. All members output a prediction, and a meta-learner makes a final decision based on these predictions. *Cascade Generalization* [53] imposes an order to the ensemble members. Each member makes a prediction also based on the prediction of the previous members. Empirical evidence suggests that Cascade Generalization performs better than vanilla Stacking. Caruana et al. [30] propose a hill-climbing method to select an appropriate set of base-learners from a large library of models.

3.5 Conservation for Generalization Performance

The 'Law of Conservation for Generalization Performance' (also known as the No Free Lunch Theorem), states that when taken across all learning tasks, all learning algorithms perform equally well [134]. Basically, it states that for each algorithm there are datasets on which it performs well, and there are datasets on which it performs badly; each algorithm makes its own assumptions about the data.

This theorem can be illustrated by means of the following example. Consider a binary dataset d on which a deterministic algorithm α obtains a certain predictive accuracy $y(\alpha, d)$ on a test set of unseen examples. In the universe of all imaginable datasets, there exists a dataset $\hat{d}$ that has the exact same training instances, yet all class labels of the unseen test set are exactly opposite, leading to a predictive accuracy of $y(\alpha, \hat{d})$. It is easy to see that in this example $y(\alpha, d) + y(\alpha, \hat{d}) = 1$. A generalization leads to the conclusion that taken over all existing datasets, the performance of all deterministic algorithms is exactly the same.

It might seem that this theoretical result subverts the purpose of meta-learning. If all algorithms perform equally, building a meta-algorithm that overcomes this limitation would be paradoxical. However, Giraud-Carrier and Provost [60] point out that under a reasonable assumption, the 'Law of Conservation for Generalization Performance' is irrelevant to Machine Learning research.

They define *the weak assumption of Machine Learning* to be that the process that presents us with learning problems induces a non-uniform probability distribution over the possible functions. In other words, among all possible learning tasks, some are more probable to occur than others. This assumption is widely accepted; without accepting this assumption, all Machine Learning research would be pointless, as there would be no reason to believe that the learned models generalize beyond the dataset. Under this assumption, meta-learning does not violate the law of conservation for generalization performance.

Furthermore, they define the *the strong assumption of Machine Learning* that the probability distribution of these functions is known implicitly or explicitly, at least to a useful approximation. In some cases, even this assumption is reasonable. A good meta-learning system selects algorithms that work well on the tasks that are probable to occur in the problem domain of interest. A meta-learning system based on knowledge from one domain of tasks can not be expected to make accurate recommendations for tasks from another domain.

The most important contribution of the 'Law of Conservation for Generalization Performance' is that it shows that learning comes with assumptions. The stronger the assumptions we make, the more efficient a learning procedure can be. Meta-learning relies on the same assumptions as any other learning procedure does.

Table 3.1: Some characteristics about different models. Taken from [65].

Characteristic	Neural Networks	SVM	Trees	MARS	k-NN, Kernels
Natural handling of mixed data types	-	-	+	+	-
Handling of missing values	-	-	+	+	+
Robustness to outliers in input space	-	-	+	-	+
Insensitive to monotone transformations of inputs	-	-	+	-	-
Computational Scalability (many instances)	-	-	+	+	-
Ability to deal with irrelevant inputs	-	-	+	+	-
Ability to extract linear combinations of features	+	+	-	-	+/-
Interpretability	-	-	+/-	+	-
Predictive Power	+	+	-	+/-	+

3.6 Model Characteristics

Although applications to the algorithm selection problem (Chapter 3.2–3.4) are useful in their own right and lead to various valuable insights, a disadvantage is that they typically focus on a small set of problems, and that it is hard to interpret and generalize the results. On the other side of the spectrum, Hastie et al. [65] composed a set of simple characteristics, describing strong and weak points of models. These are shown in Table 3.1. Most of these models are already described in Chapter 2.4. MARS [49] is a regression model based on *splines*, a mathematical function that consist of various polynomials at different input domains.

Most of the characteristics actually make a lot of sense. For example, it followed already from the model description in Chapter 2.4 that Logistic Regression, Support Vector Machines and Neural Networks do not handle missing values natively, as it would remove one part of the equation. Furthermore, trees and splines are typically considered quite interpretable, even though this is a subjective matter. This kind of simple characteristics already have great value. It summarizes the strong and weak points of various models, and gives domain experts that use machine learning a good overview of what kind of models they should use for their problem.

The mentioned characteristics also have shortcomings. For example, the results

are presented without any discussion or evidence. Although most of the statements seem intuitively correct, there are also some controversies. For example, it is widely considered that decision trees have a high ability to deal with irrelevant inputs. However, experimental results suggest that the truth is a bit more subtle [111]. Also, the low robustness to outliers in input space of Support Vector Machines is debatable.

This table immediately exploits a huge problem in current Machine Learning literature. New models are typically evaluated solely based on predictive power; sometimes interpretability and computational scalability are also taken into account. However, every new method should be evaluated on more than just these criteria. Of course, a method does not have to excel on all of the characteristics. As can be seen, none of the currently characterised models do. Extending this table to additional models, algorithms and characteristics is a very useful form of meta-learning.

3.7 Bias Variance Profile

A common problem of modelling is *over-fitting*, a phenomenon where irrelevant relationships between the data and the class are being encoded. This typically happens when there is too little data, there is too much noise in the data or the model is too complex. When a model performs well on the training set and mediocre on the test set, it is likely that it over-fitted the training data. One way of analysing and understanding this better, is by means of a *bias variance decomposition* [82]. Kohavi et al. [82] define three types of errors, *bias* errors, *variance* errors and *irreducible errors*. The irreducible errors are the errors that can not be avoided by any learner, e.g., when the test set contains instances with the same attribute values, yet different labels. Variance is the tendency to learn random things irrespective of the real function (it hallucinates patterns). This often happens when a model is too complex, and models dependencies that do not exist in the real world; it over-fits the data. Bias is the tendency to consistently learn the same wrong thing. This often happens when a model is too simple, and unable to model the true relationship between the data and the class; it *under-fits* the data. Bias and variance are often resembled by throwing darts at a board (see Figure 3.3). A model that only makes irreducible errors is called a *Bayes-optimal* model.

Many methods attempt to prevent over-fitting. One way of doing so is adding a *regularization component*, penalizing complex models, and therefore favouring simpler models with less room to over-fit. However, by imposing this preference towards simpler models, the solution is exposed to the other type of error, i.e., under-fitting. A notable exception is Bagging, a technique that reduces the risk of over-fitting with slightly increasing the bias (Chapter 3.4). Dealing with bias and variance is essentially

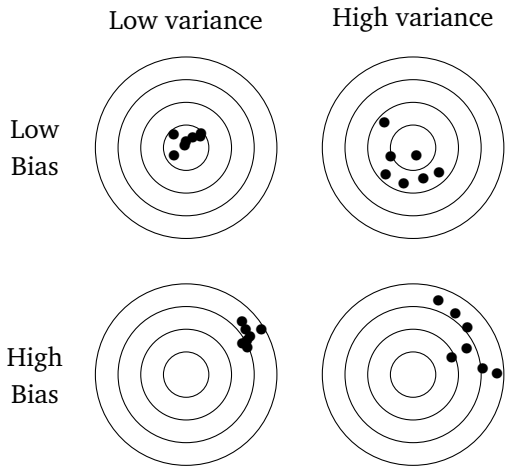


Figure 3.3: Bias and Variance. Image adapted from [38].

about finding a trade-off between the possibility of over-fitting and under-fitting. As additional components are added to the model, the model complexity rises, the bias reduces and the variance increases. In other words, bias has a negative first-order derivative in response to model complexity while variance has a positive slope. The task of a Machine Learning practitioner is to find a good trade-off between the two.

Although in general Machine Learning applications the total number of errors matters, understanding bias and variance gives insight in the predictive behaviour of learning algorithms. Therefore, it is a very important aspect of meta-learning, potentially increasing both our knowledge about algorithms and the performance of newly created algorithms.

3.8 Discussion

This chapter reviewed aspects of meta-learning; techniques that improve knowledge of the learning process and techniques that help selecting an appropriate learning algorithm. The algorithm selection problem can be solved by learning techniques as well as search techniques. Often, the decision of which paradigm to chose comes down to whether experimentation with the data is allowed. When the data is not accessible yet or a recommendation must be provided relatively fast, the search paradigm is not applicable; hence the learning paradigm should be preferred. On the other hand, when there is time to try multiple configurations, the search paradigm is likely to recommend a good algorithm and parameter setting combination. Sometimes, the

search methods can be guided by some form of meta-learning.

The advantages of the learning approaches are plenty. First, the resulting meta-model is very general; it can be applied to all unseen problems. Second, algorithm selection for new tasks is rather inexpensive. Although building the meta-model can be a computational intensive task, applying it to new tasks happens typically fast. Finally, it is reasonable to expect the first attempted solution already to yield good result. Apart from calculating the meta-features, there are no start-up costs.

However, this approach also has some intrinsic limitations. First, it is hard to construct a meta-feature set that adequately characterizes the problem space [86]. Second, the most successful meta-features, landmarkers, can be computationally expensive, limiting the options [108]. Finally, because not all classifiers run on all datasets, or take prohibitively long to do so, the meta-dataset usually contains many missing values, complicating the classification task.

The search paradigm can be seen as a more thorough alternative. First, as it naturally tries multiple promising candidates, it will return a good solution with high probability. In the multi-armed bandit approach, there are even theoretical guarantees about the performance. Second, it is an iterative approach, that is constantly expected to improve itself. After a few iterations we can already expect reasonable recommendations, and it is likely that these keep on improving. Third, most search methods do not require a meta-dataset. As it is considered a laborious task to create an appropriate meta-dataset, it is convenient that search methods do not require such.

There are also limitations, compared to the learning paradigm. Most prominently, it comes with additional run time. Search methods are built upon function evaluations, i.e., multiple models are built and evaluated. Furthermore, obtained results do not generalize. The search procedure needs to be executed for each task again. Nothing is learned about the interaction between the mechanism of learning and the concrete contexts in which that mechanism is applicable. In that sense, this form of algorithm selection does not obey the definition of meta-learning given by Vilalta et al. [159].

There are some additional challenges that meta-learners are faced with. Many algorithm selection problems are subject to the curse of dimensionality [152]. Various studies proposed many different types of meta-features, yet the amount of available datasets is relatively small. Although many datasets exists, most of these are in people's labs and heads, not available to the community. Furthermore, there is a high computational cost for each instance. Indeed, all algorithms should be ran on it, which takes a lot of resources. For these reasons, experiment databases have been proposed [124, 153, 154, 155]. These aim to store and organise datasets and results from earlier experiments, available to the whole community. In the next chapter we

will review how Machine Learning and meta-learning research can benefit from such infrastructures.

4 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 landmarkers. 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 build 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) hyper-parameter 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 website. 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

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

Figure 4.3: Example of an OpenML run.

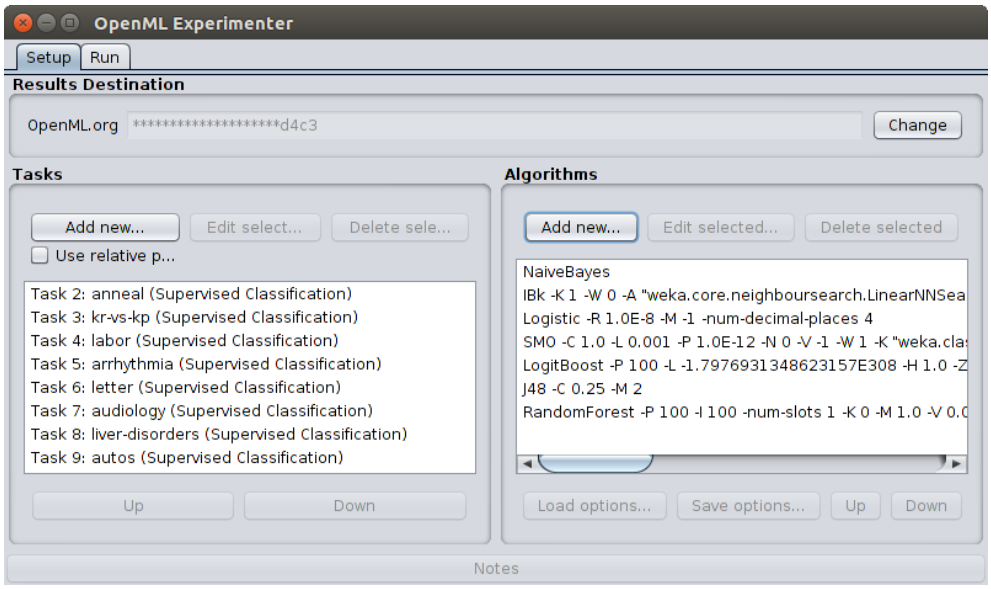
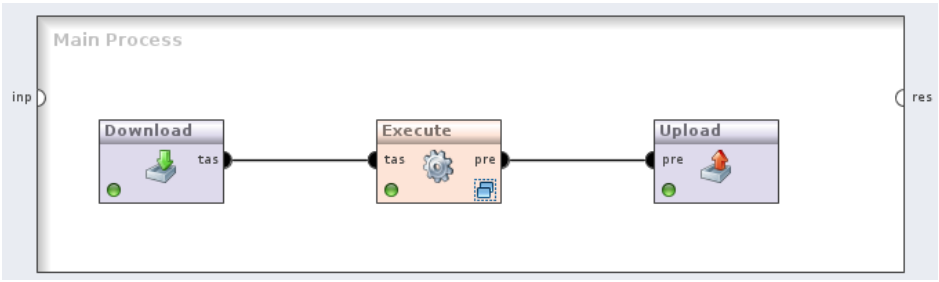


Figure 4.4: WEKA integration of OpenML.

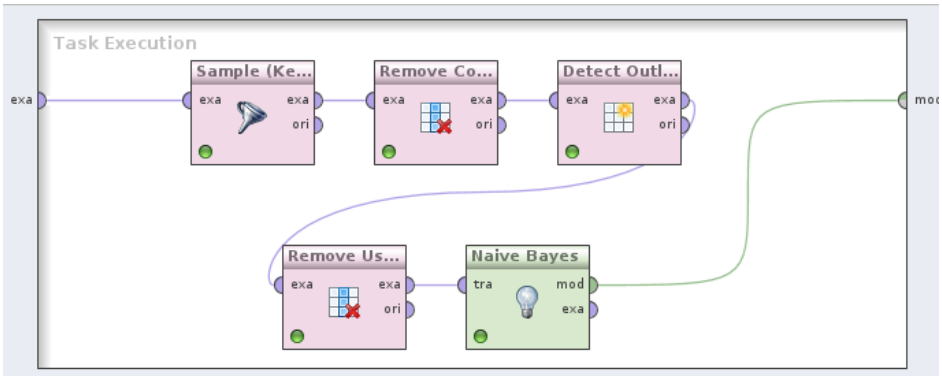
training set, and produces a model. This model is then used to predict the labels for the observations in the associated test set. An example of such a sub-workflow, including several pre-processing steps, is shown in Figure 4.5b. The output of this super-operator is a data structure containing predictions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

```
library(mlr)
library(OpenML)

# set API key to read only key (replace it with your own key)
setOMLConfig(apikey = "b2994bdb7ecb3c6f3c8f3b35f4b47f1f")

lrn = makeLearner("classif.randomForest")

# upload the new flow (with information about the algorithm and settings);
# if this algorithm already exists on the server, one will receive a message
# with the ID of the existing flow
flow.id = uploadOMLFlow(lrn)

# the last step is to perform a run and upload the results
run.mlr = runTaskMlr(task, lrn)
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

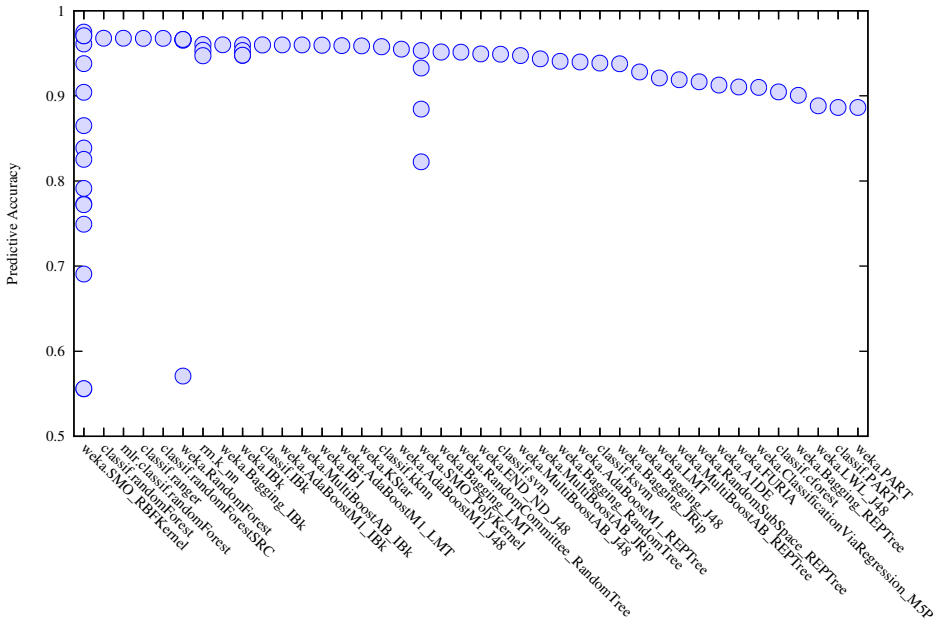


Figure 4.7: Performance of various algorithms on the 'letter' dataset.

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 RapidMiner. 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
k-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

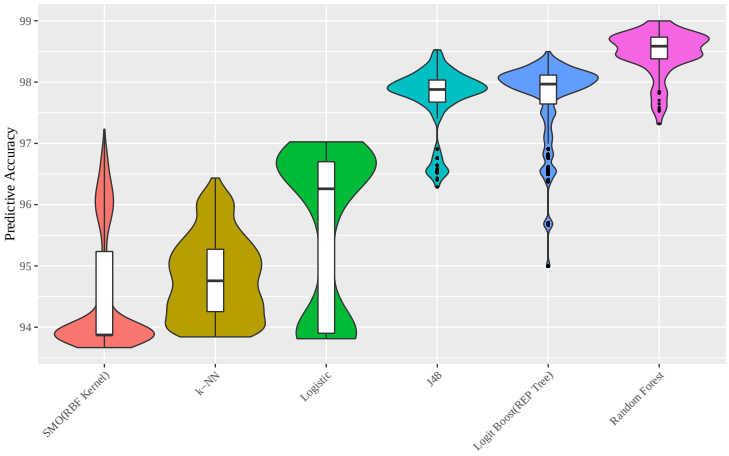


Figure 4.8: Variation in predictive accuracy when optimizing parameters on the "sick" dataset.

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

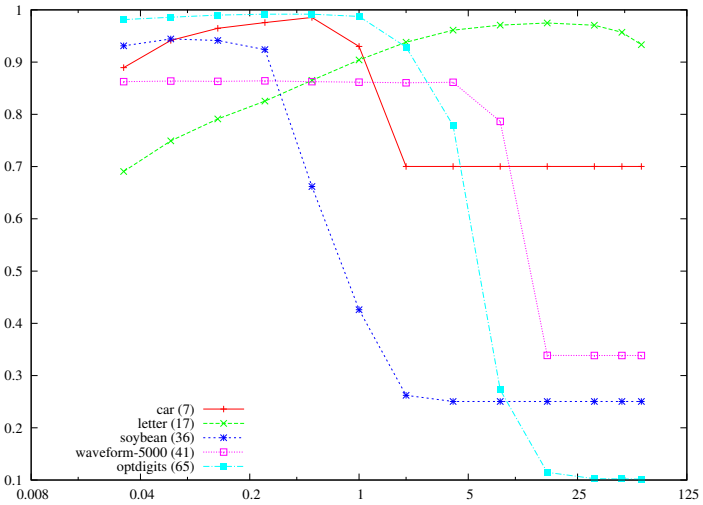


Figure 4.9: Effect of gamma parameter of the RBF kernel for Support Vector Machines on various datasets. The number between brackets indicates the number of attributes of such dataset.

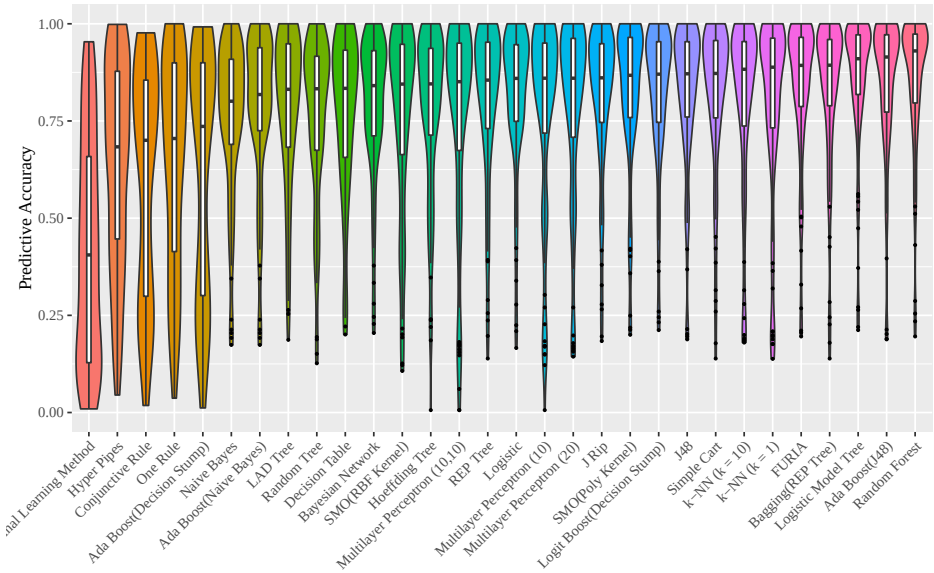
parameter is different for all datasets, as would be expected. It seems that setting this value too low is less harmful for performance than setting it too high. All-together, the parameter landscape seems to be smooth, i.e., there are no spikes in the plots.

4.5.1.4 Comparison over many datasets

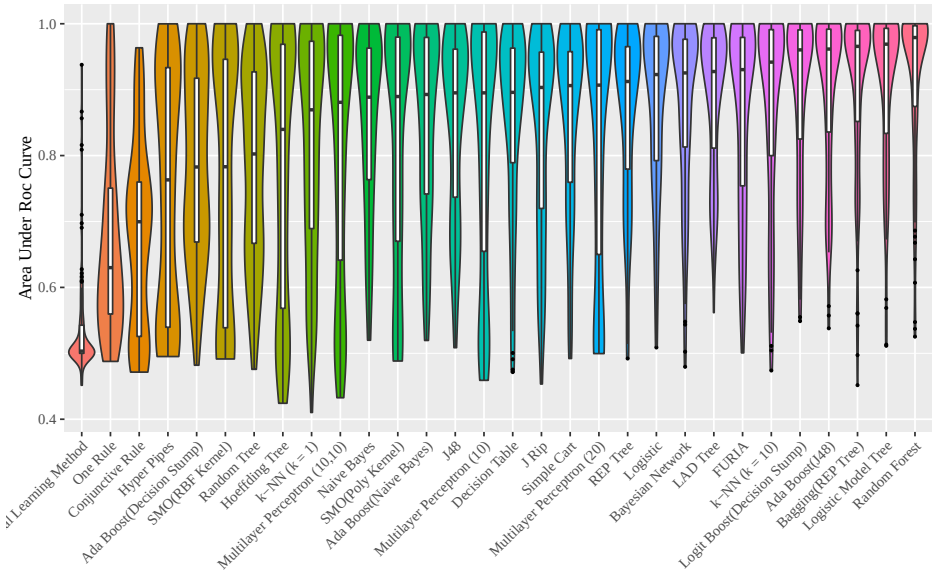
In order to gain a decent understanding of how a classifier performs, it could be evaluated over a wide range of datasets. We selected a set of classifiers and datasets from OpenML. All classifiers were ran on all datasets.

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.

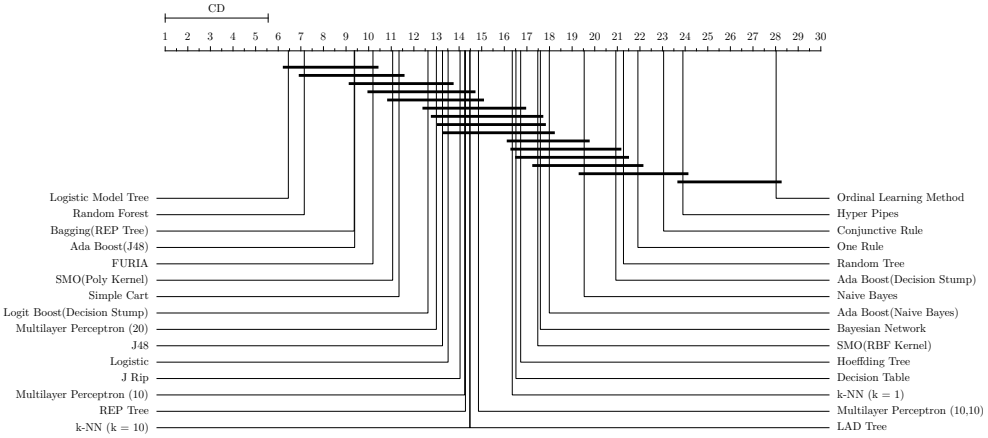


Figure 4.11: Results of Nemenyi test ($\alpha = 0.05$) on the predictive accuracy of classifiers in OpenML. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

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

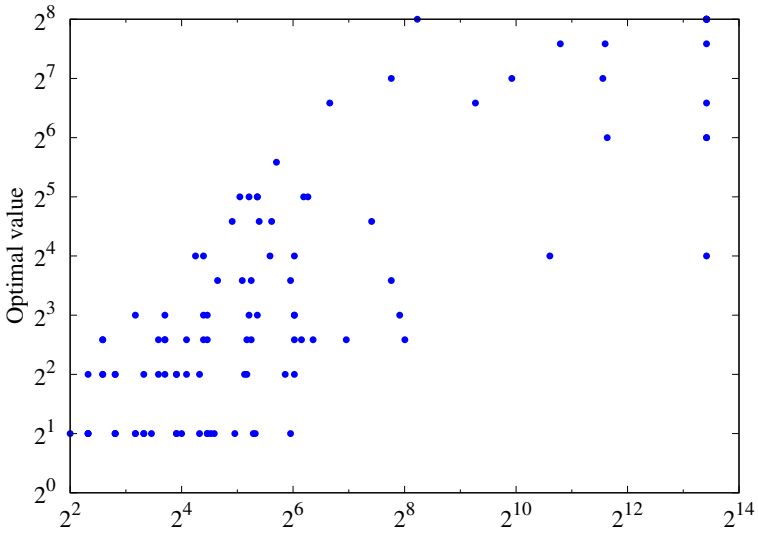
As OpenML contains many runs of algorithms with different parameter settings, we can use these results to gain more insight in the interplay between a certain data characteristic and the optimal value of such parameter. In order to do so, we fix all parameters to a given value (e.g., the default value) 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.

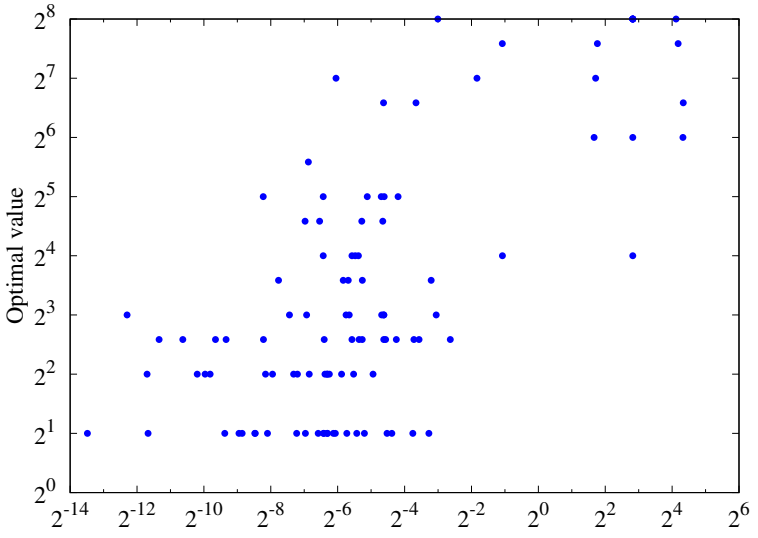
The plot seems to confirm some sort of relation between the number of attributes and the optimal value for this parameter. However, these scatter-plots also have some intrinsic disadvantages: overlapping points are not visible as such, it only shows the absolute best value of the parameter and the results 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

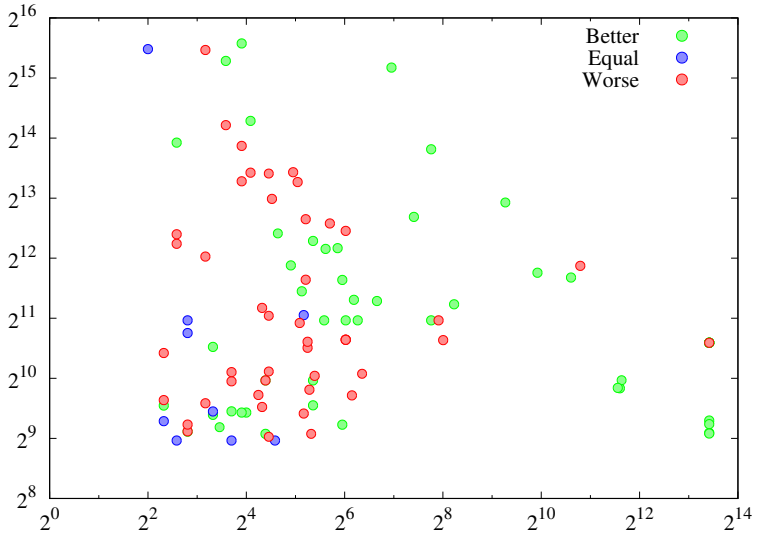
Figure 4.12: The optimal values of the 'Random Features per Split' parameter of Random Forests

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.

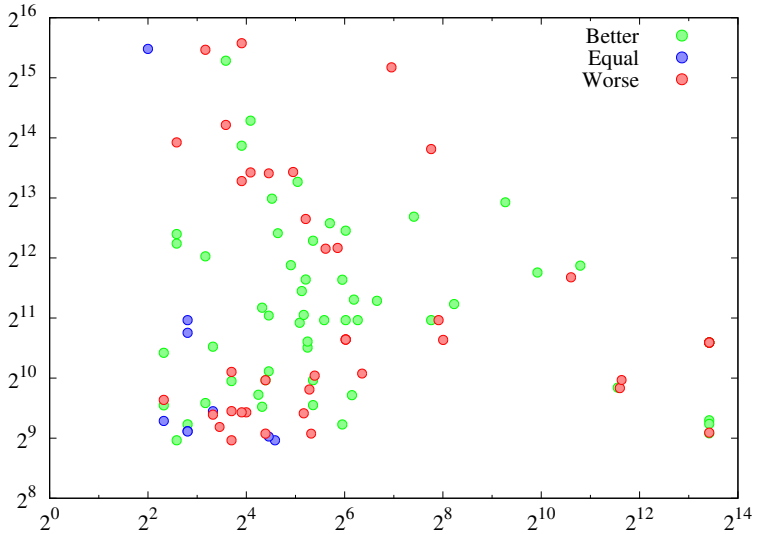
Many OpenML tasks have for a given classifier results on how it performed with and without various pre-processing operators. For example, in Weka, we can use the Feature Selected Classifier, to apply feature selection on a given dataset. We simply combine the results of that algorithm with and without 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.

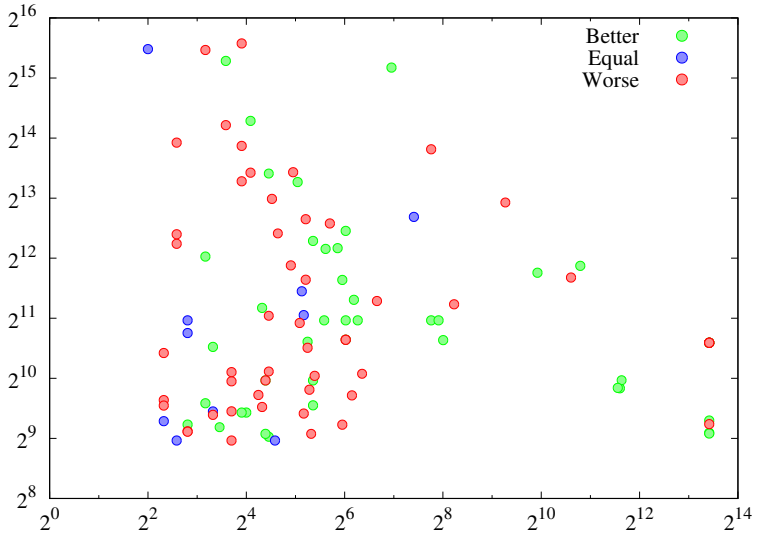


(a) k-NN

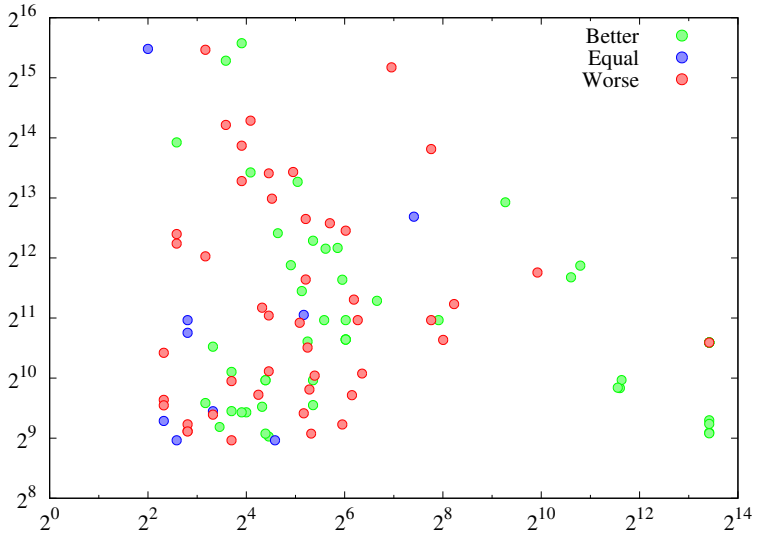


(b) Naive Bayes

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

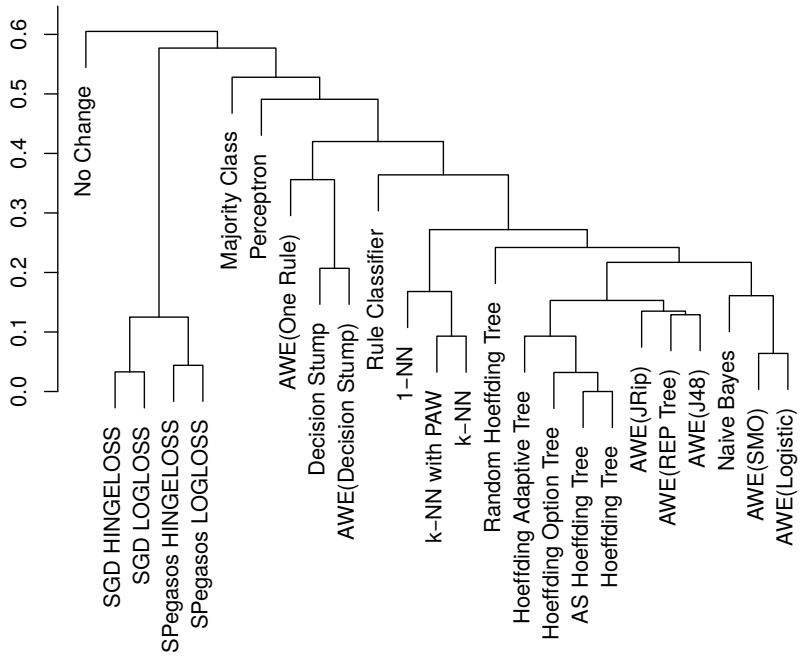


Figure 4.14: Hierarchical clustering of stream classifiers, averaged over 51 data streams from OpenML.

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.

5 Data Streams

Real-time analysis of data streams is a key area of data mining research. Many real world collected data are in fact streams where observations come in one by one, and algorithms processing these are often subject to time and memory constraints. This chapter focusses on ensembles for data streams. Ensembles of classifiers are among the best performing classifiers available in many data mining applications, including the mining of data streams. Rather than training one classifier, multiple classifiers are trained, and their predictions are combined according to a given voting schedule. An important prerequisite for ensembles to be successful is that the individual models are diverse. One way to vastly increase the diversity among the models is to build an *heterogeneous* ensemble, comprised of fundamentally different model types. However, most ensembles developed specifically for the dynamic data stream setting rely on only one type of base-level classifier, most often Hoeffding Trees. This chapter focusses on meta-learning techniques to combine the intrinsically different models to heterogeneous ensembles.

OpenML played a vital role in the development of these techniques. In order to build a good heterogeneous ensemble technique, two questions need to be addressed: which classifiers to use and how to weight their individual votes. Having stored all experimental results of possible base-classifiers, it becomes a matter of database queries to test certain combinations. Classifiers that appear to complement each other well, can then be combined into a heterogeneous ensemble. Furthermore, most data stream research involves only few streams. Having a large collection of datasets and data streams, OpenML is an indispensable factor in scaling up this direction of research. Finally, all these results organised together foster new research questions and discoveries; Chapter 5.6 shows an example of this. OpenML is in this sense the open

experiment database of data stream research.

5.1 Introduction

Modern society produces vast amounts of data coming, for instance, from sensor networks and various text sources on the internet. Much of this data is in fact a stream where observations come in one by one. Real-time analysis of such data streams is a key area of data mining research, typically the algorithms processing these are subject to time and memory constraints. The research community developed a large number of machine learning algorithms capable of online modelling general trends in stream data and make accurate predictions for future observations. In many applications, *ensembles* of classifiers are the most accurate classifiers available. Rather than building one model, a variety of models are generated that all vote for a certain class label.

One way to vastly improve the performance of ensembles is to build *heterogeneous* ensembles, consisting of models generated by different techniques, rather than *homogeneous* ensembles, in which all models are generated by the same technique. One technique to build such a heterogeneous ensemble is Stacking [53, 163]. It has been extensively analysed in classical batch data mining applications. As the underlying techniques upon which Stacking relies can not be trivially transferred to the data stream setting, there are currently no successful heterogeneous ensemble techniques in the data stream setting. State of the art heterogeneous ensembles in a data stream setting typically rely on meta-learning [125, 133]. These approaches both require the extraction of computationally expensive meta-features and yield marginal improvements.

In this work we introduce an elegant technique that natively combines heterogeneous models in the data stream setting. As data streams are constantly subject to change over time, the most accurate classifier for a given interval of observations also changes frequently, as illustrated by Figure 5.1. We propose a way to measure the performance of ensemble members on recent observations and combine their votes based on this.

This chapter introduces the following concepts. First, we describe Online Performance Estimation, a framework that provides *dynamic* weighting of the votes of individual ensemble members across the stream [128]. Utilizing this framework, we introduce a new ensemble technique that combines heterogeneous models. The online performance estimation framework can also be used to generate traditional meta-features, which can also be used in a predictive manner. Finally, an experiment covering 60 data streams shows that both techniques are competitive with state of the art ensembles, while requiring significantly less run time.

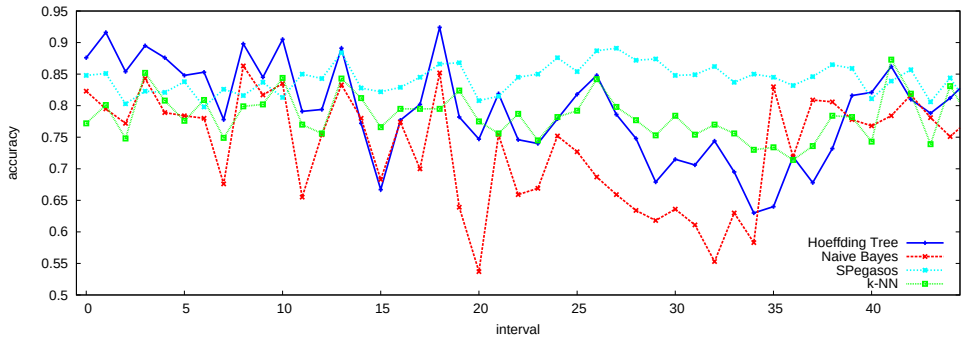


Figure 5.1: Performance of four classifiers on intervals (size $1{,}000$) of the 'electricity' dataset. Each data point represents the accuracy of a classifier on the most recent interval.

The remainder of this chapter is organised as follows. Chapter 5.2 surveys data streams, and Chapter 5.3 introduces the proposed methods. Chapter 5.4 and Chapter 5.5 describe various experiments, covering the performance of the proposed methods and the relevance of online performance estimation. Chapter 5.6 describes an interesting pattern that was discovered by mining the experiment repository. Chapter 5.7 concludes.

5.2 Related Work

It has been widely recognized that data stream mining differs significantly from conventional batch data mining [13, 14, 40, 57, 118]. In the conventional batch setting, a finite amount of stationary data is provided and the goal is to build a model that fits the data as well as possible. When working with data streams, we should expect an infinite amount of data, where observations come in one by one and are being processed in that order. Furthermore, the nature of the data can change over time, known as *concept drift*. Classifiers (and other modelling techniques) operating on streams of data should be able to detect when a learned model becomes obsolete and update it accordingly.

Common Approaches. Some batch classifiers can be trivially adapted to a data stream setting. Examples are k Nearest Neighbour [10, 165], Stochastic Gradient Descent [18] and SPegasos (Stochastic Primal Estimated sub-GrAdient SOlver for SVMs) [139]. Both Stochastic Gradient Descent and SPegasos are gradient descent methods, capable of learning a variety of linear models, such as Support Vector Ma-

chines and Logistic Regression, depending on the chosen loss function.

Other classifiers have been created specifically to operate on data streams. Most notably, the Hoeffding Tree induction algorithm, which inspects every example only once, and stores per-leaf statistics to calculate the *information gain* on which the split criterion is determined [39]. The *Hoeffding bound* states that the true mean of a random variable of a given range will not differ from the estimated mean by more than a certain value. This provides statistical evidence that a certain split is superior over others. As Hoeffding Trees seem to work very well in practice, many variants have been proposed, such as Hoeffding Option Trees [109], Adaptive Hoeffding Trees [12] and Random Hoeffding Trees [15].

Finally, a commonly used technique to adapt traditional batch classifiers to the data stream setting is training them on a window of w recent examples: after w new examples have been observed, a new model is built. This approach has the advantage that old examples are ignored, providing natural protection against concept drift. A disadvantage is that it does not operate directly on the most recently observed data, not before w new observations are made and the model is retrained. Read et al. [118] compare the performance of these *batch-incremental* classifiers with common data stream classifiers, and conclude that the overall performance is equivalent, although the batch-incremental classifiers generally use more resources (time and memory).

Ensembles. Ensemble techniques train multiple classifiers on a set of weighted training examples, and these weights can vary for different classifiers. In order to classify test examples, all individual models produce a prediction, also called a *vote*, and the final prediction is made according to a predefined voting schema, e.g., the class with the most votes is selected.

Bagging [23] exploits the instability of classifiers by training them on different *bootstrap replicates*: resamplings (with replacement) of the training set. Effectively, the training sets for various classifiers differ by the weights of their training examples. Online Bagging [101] operates on data streams by drawing the weight of each example from a $Poisson(1)$ distribution, which converges to the behaviour of the classical Bagging algorithm if the number of examples is large. As the Hoeffding bound gives statistical evidence that a certain split criteria is optimal, this makes them more stable and hence less suitable for use in a Bagging scheme. However, empirical evaluation suggests that this yields good results nonetheless.

Boosting [135] is a technique that sequentially trains multiple classifiers, in which more weight is given to examples that where misclassified by earlier classifiers. Online Boosting [101] applies this technique on data streams by assigning more weight to training examples that were misclassified by previously trained classifiers in the ensemble. The Accuracy Weighted Ensemble (AWE) works well in combination with batch-incremental classifiers [161]. It maintains multiple recent models trained on

different batches of the data stream, and weights the votes of each model based on the expected predictive accuracy.

Concept drift. One property of data streams is that the underlying concept that is being learned can change over time (e.g., [161]). This is called concept drift. Some of the aforementioned methods naturally deal with concept drift. For instance, k Nearest Neighbour maintains a number of w recent examples, substituting each example after w new examples have been observed. *Change detectors*, such as Drift Detection Method (DDM) [55] and Adaptive Sliding Window Algorithm (ADWIN) [11] are stand-alone techniques that detect concept drift and can be used in combination with any stream classifier. Both rely on the assumption that classifiers improve (or at least maintain) their accuracy when trained on more data. When the accuracy of a classifier drops in respect to a reference window, this could mean that the learned concept is outdated, and a new classifier should be built. The main difference between DDM and ADWIN is the way they select the reference window. Furthermore, classifiers can have built-in drift detectors. For instance, Ultra Fast Forest of Trees [56] are Hoeffding Trees with a built-in change detector for every node. When an earlier made split turns out to be obsolete, a new split can be generated.

It has been recognised that some classifiers recover faster from sudden changes of concepts than others. The *recovery analysis* framework measures the ability of classifiers to recover from concept drift [138]. It distinguishes instance-based classifiers that operate directly on the data (e.g., k-NN) and model-based classifiers, that build and maintain a model (e.g., tree algorithms, fuzzy systems). Their experimental results suggest, quite naturally, that instance-based classifiers generally have a higher capability to recover from concept drift than model-based classifiers.

Evaluation. As data from streams is non-stationary, the well-known cross-validation procedure for estimating model performance is not suitable. A commonly accepted estimation procedure is the *prequential method* [57], in which each example is first used to test the current model, and afterwards (either directly after testing or after a delay) becomes available for training. An advantage of this method is that it is tested on all data, and therefore no specific holdout set is needed.

Meta-Learning. Meta-learning approaches on data streams often train multiple base-classifiers and a meta-model decides, for each data point, which of the base-learners will make a prediction. Meta-knowledge can be obtained from two sources. The first option is to extract meta-knowledge obtained from earlier in the stream [133]. One advantage is that there is no representative meta-dataset required: we can assume that the data from earlier in the stream is adequate. The disadvantage is that it requires more run time, as a meta-model needs to be trained while processing the stream. The other option is to extract meta-knowledge from other data streams [125, 126, 128]. The advantage is that this is relatively fast: the meta-model can be trained

a priori and does not require additional resources while processing the stream.

Another technique uses meta-learning on time series with recurrent concepts: when concept drift is detected, a meta-learning algorithm decides whether a model trained previously on the same stream could be reused, or whether the data is so different from before that a new model must be trained [54]. Nguyen et al. [94] propose a method that combines feature selection and heterogeneous ensembles; members that perform poorly according to a drift detector can be replaced.

5.3 Methods

Traditional Machine Learning problems consist of a number of *examples* that are observed in arbitrary order. In this work we consider classification problems. Each example $e = (\mathbf{x}, l(\mathbf{x}))$ is a tuple of p predictive attributes $\mathbf{x} = (x_1, \ldots, x_p)$ and a target attribute $l(\mathbf{x})$. A data set is an (unordered) set of such examples. The goal is to approximate a labelling function $l : \mathbf{x} \rightarrow l(\mathbf{x})$. In the data stream setting the examples are observed in a given order, therefore each data stream S is a sequence of examples $S = (e_1, e_2, e_3, \ldots, e_n, \ldots)$, where n is the number of observations (possibly infinite). Consequently, S_i refers to the i^{th} example in data stream S. The set of predictive attributes of that example is denoted by PS_i, likewise $l(PS_i)$ maps to the corresponding label. Furthermore, the labelling function that needs to be learned can change over time due to concept drift. When applying an ensemble of classifiers, the most relevant variables are which base-classifiers (members) to use and how to weight their individual votes.

5.3.1 Online Performance Estimation

In most common ensemble approaches, all base-classifiers are given the same weight (as done in Bagging and Boosting) or their predictions are otherwise combined to optimise the overall performance of the ensemble (as done in Stacking). An important property of the data stream setting has not been taken into account: due to the possible occurrence of concept drift it is likely that in most cases recent examples are more relevant than older ones. Moreover, due to the fact that there is a temporal component in the data, we can actually measure how ensemble members have performed on recent examples, and adjust their weight in the voting accordingly. In order to estimate the performance of a classifier on recent data, van Rijn et al. [127] proposed:

$$P_{win}(l', c, w, L) = 1 - \sum_{i=max(1,c-w)}^{c-1} \frac{L(l'(PS_i), l(PS_i))}{min(w, c-w)} \tag{5.1}$$

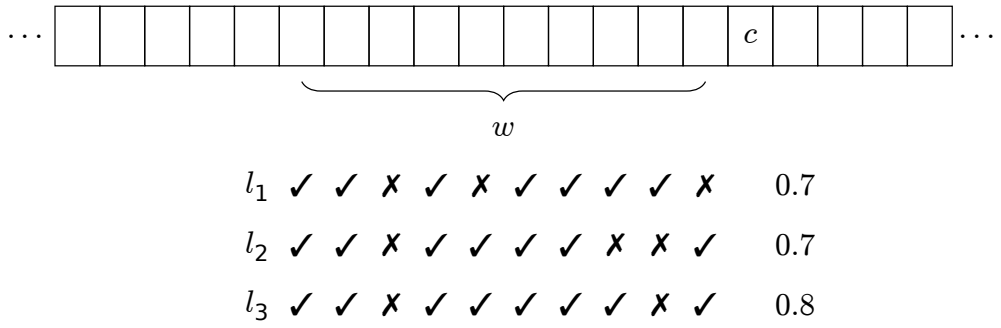


Figure 5.2: Schematic view of Windowed Performance Estimation. For all classifiers, w flags are stored, each flag indicating whether it predicted a recent observation correctly.

where l' is the learned labelling function of an ensemble member, c is the index of the example we want to predict and w is the number of training examples over which we want to estimate the performance of ensemble members. Finally, L is a loss function that compares the labels predicted by the ensemble member to the true labels. The most simple version is a zero/one loss function, which returns 0 when the predicted label is correct and 1 otherwise. More complicated loss functions can also be incorporated. The outcome of P_{win} is in the range $[0, 1]$, with better performing classifiers obtaining a higher score. The performance estimates for the ensemble members can be converted into a weight for their votes, at various points over the stream. For instance, the best performing members at that point could receive the highest weights. Figure 5.2 illustrates this.

There are a few drawbacks to this approach. Firstly, it requires the ensemble to store the $w \times n$ additional values, which is inconvenient in a data stream setting, where both time and memory are important factors. Secondly, it requires the user to tune a parameter which highly influences performance. Lastly, there is a hard cut-off point, i.e., an observation is either in or out of the window. What we would rather model is that the most recent observations are given most weight, and gradually lower this for less recent observations.

In order to address these issues, *fading factors* (further described in, e.g., Gama et al. [58]) can be used as an alternative. Fading factors give a high importance to recent predictions, whereas the importance fades away when they become older. This is illustrated by Figure 5.3. The red (solid) line shows a relatively fast fading factor, where the effect of a given prediction has already faded away almost completely after 500 new observations, whereas the blue (dashed) line shows a relatively slow fading

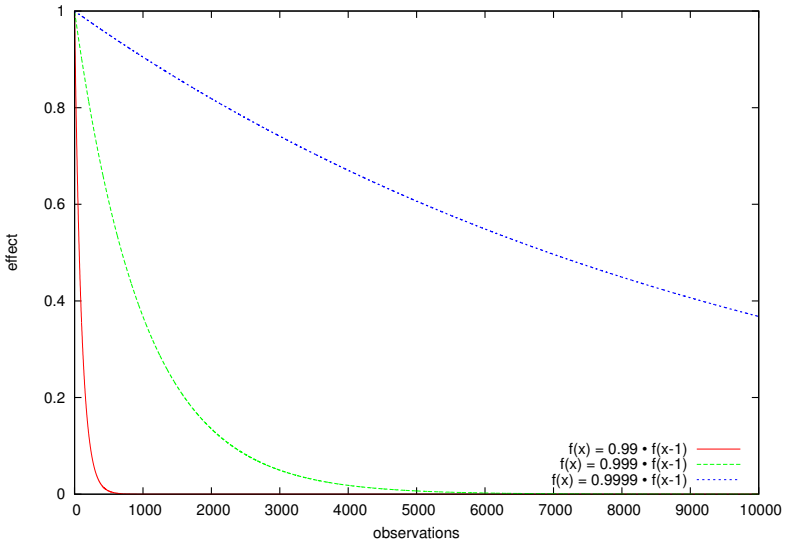
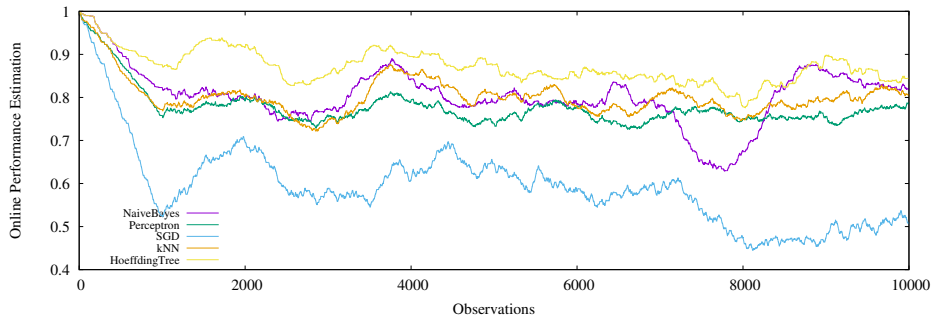


Figure 5.3: The effect of a prediction after a number of observations, relative to when it was first observed (for various values of α).

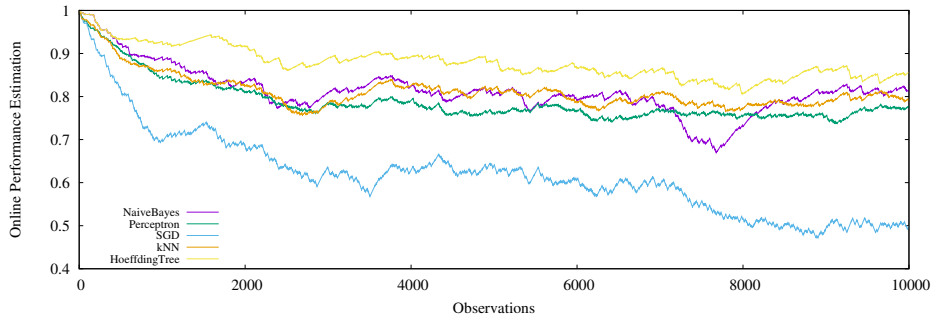
factor, where the effect of an observation is still considerably high, even when $10{,}000$ observations have passed in the meantime. Note that even though all these functions start at 1, in practise we need to scale this down to $1 - \alpha$, in order to constrict the complete function within the range $[0, 1]$. Putting this all together, we get:

$$P(l', c, \alpha, L) = \begin{cases} 1 & \text{iff } c = 0 \\ P(l', c-1, \alpha, L) \cdot \alpha + (1 - L(l'(PS_c), l(PS_c))) \cdot (1 - \alpha) & \text{otherwise} \end{cases} \tag{5.2}$$

where, similar to Eq. 5.1, l' is the learned labelling function of an ensemble member, c is the index of the last seen training example and L is a loss function that compares the labels predicted by the ensemble member to the true labels. Fading factor α (range $[0, 1]$) determines at what rate historic performance becomes irrelevant, and is to be tuned by the user. A value close to 0 will allow for rapid changes in estimated performance, whereas a value close to 1 will keep them rather stable. The outcome of P is in the range $[0, 1]$, with better performing classifiers obtaining a higher score. In Chapter 5.5 we will see that the fading factor parameter is more robust and easier to tune than the window size parameter. When building an ensemble based upon Online Performance Estimation, we can now choose between a Windowed approach (Eq. 5.1) and Fading Factors (Eq. 5.2). Figure 5.4 shows how the estimated performance for each base-classifier evolves on the start of the electricity data stream. Both



(a) Windowed, window size 1000



(b) Fading Factors, $\alpha = 0.999$

Figure 5.4: The estimated performance of each algorithm given previous examples, measured on the start of the electricity data stream.

figures expose similar trends: apparently, on this data stream the Hoeffding Tree classifier performs best and the Stochastic Gradient Descent algorithm performs worst. However, both approaches differ subtly in the way the performance of individual classifiers are measured. The Windowed approach contains many spikes, whereas the Fading Factor approach seems more stable.

5.3.2 Ensemble Composition

In order for an ensemble to be successful, the individual classifiers should be both accurate and diverse [64]. When employing a homogeneous ensemble, choosing an appropriate base-learner is an important decision. For heterogeneous ensembles this is even more true, as we have to choose a set of base-learners.

Figure 5.5 shows some basic performance results of the classifiers on 60 data streams. Figure 5.5a shows a violin plot of the performance of all classifiers, with

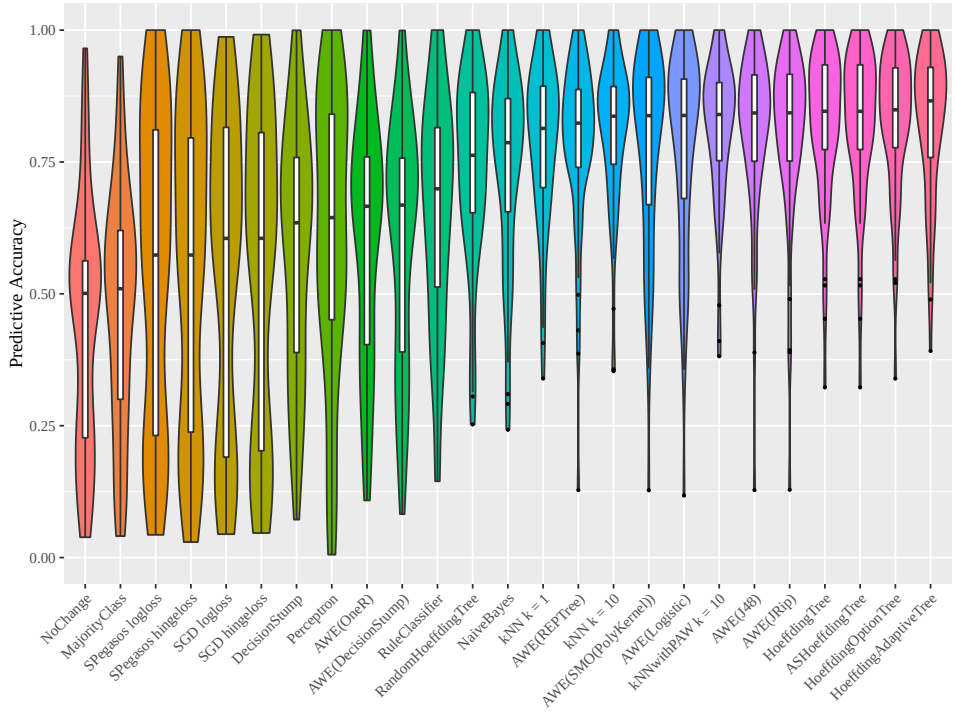
a box plot in the middle. The classifiers are sorted by median accuracy. Two common Data Stream baseline methods, the No Change classifier and the Majority Class classifier, end at the bottom of the ranking based on accuracy. This indicates that most of the selected data streams are both balanced (in terms of class labels) and do not have high auto-correlation. In general, tree-based methods seem to perform best.

Figure 5.5b shows the result of a statistical test on the base-classifiers. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent. The results confirm some of the observations made based on the violin plots, e.g., the baseline models (Majority Class and No Change) perform worst; also other simple models such as the Decision Stumps and OneRule (which is essentially a Decision Stump) are inferiour to the tree-based models. Oddly enough, the instance incremental Rule Classifier does not compete at all with the Batch-incremental counterpart (AWE(JRIP)).

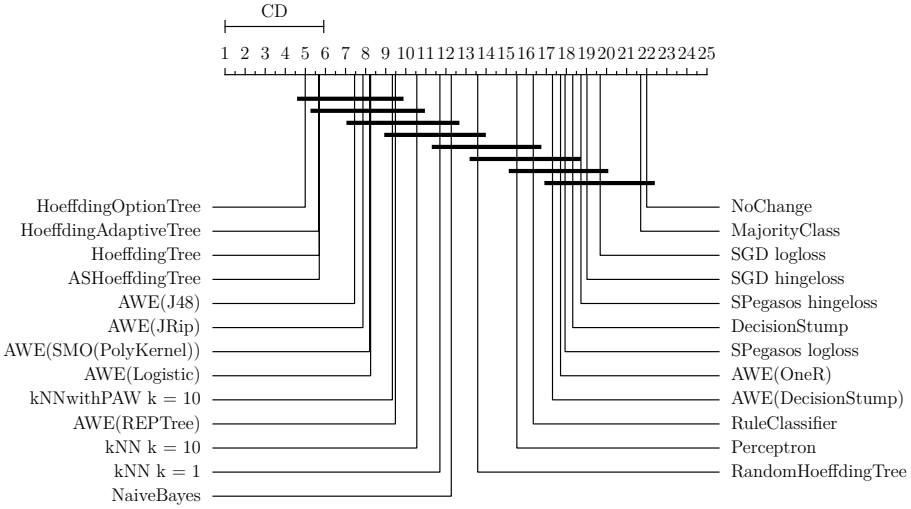
A dendrogram like the one in Figure 4.14 (page 69) can serve to select a collection of diverse and well performing ensemble members. Classifier Output Difference (COD) is a metric which measures the number of observations on which a pair of classifiers yields a different prediction [106]. This can be used to ensure diversity among the ensemble members [85]. A threshold on the Classifier Output Difference needs to be determined, selecting representative classifiers from all pairs of clusters with a distance higher than this threshold. A higher threshold would result in a smaller set of classifiers, and vice versa. For example, if we set the threshold to 0.2, we end up with an ensemble consisting of classifiers from 11 clusters. The ensemble will consist of one representative classifier from each cluster, which can be chosen based on accuracy, run time, a combination of the two (see, e.g., [21]), or any arbitrary criterion. Which exact criterion to use is outside the scope of this thesis, however in the experiments we will use a combination of accuracy and run time. Clearly, when using this technique in experiments, the dendrogram should be constructed in a leave-one-out setting: it can be created based on all data streams except for the one that is being tested.

5.3.3 BLAST

BLAST (short for **b**est **last**) is an ensemble embodying the performance estimation framework. Ideally, it consists of a group of diverse base-classifiers. These are all trained using the full set of available training observations. For every test example, it selects one of its members to make the prediction. This member is referred to as the *active classifier*. The active classifier is selected based on Online Performance Estimation: the classifier that performed best over the set of w previous trainings examples is selected as the active classifier (i.e., it gets 100% of the weight), hence the name.



(a) Performance of base-classifiers



(b) Nemenyi test, $\alpha = 0.05$

Figure 5.5: Performance of 25 data stream classifiers based on 60 data streams.

Algorithm 5.1 BLAST (Learning)

Require: Set of ensemble members M, Loss function L and Fading Factor α
1: Initialise ensemble members m_j, with $j \in \{1, 2, 3, \ldots, |M|\}$
2: Set $p_j = 1$ for all j
3: **for all** training examples $e = (\mathbf{x}, l(\mathbf{x}))$ **do**
4: **for all** $m_j \in M$ **do**
5: $l'_j(\mathbf{x}) \leftarrow$ predicted label of m_j on attributes $\mathbf{x}$ of current example e
6: $p_j \leftarrow p_j \cdot \alpha + (1 - L(l'_j(\mathbf{x}), l(\mathbf{x}))) \cdot (1 - \alpha)$
7: Update m_j with current example e
8: **end for**
9: **end for**

Formally,

$$AC_c = \arg\max_{m_j \in M} P(m_j, c-1, \alpha, L) \tag{5.3}$$

where M is the set of models generated by the ensemble members, c is the index of the current example, α is a parameter to be set by the user (fading factor) and L is a zero/one loss function, giving a penalty of 1 to all misclassified examples. Note that the performance estimation function P can be replaced by any measure. For example, if we would replace it with Equation 5.1, we would get the exact same predictions as reported by van Rijn et al. [127]. When multiple classifiers obtain the same estimated performance, any arbitrary classifier can be selected as active classifier. The details of this method are summarised in Algorithm 5.1.

Line 2 initialises a variable that keeps track of the estimated performance for each base-classifier. Everything that happens from lines 5–7 can be seen as an internal prequential evaluation method. At line 5 each training example is first used to test all individual ensemble members on. The predicted label is compared against the true label $l(\mathbf{x})$ on line 7. If it predicts the correct label then the estimated performance for this base-classifier will increase; if it predicts the wrong label, the estimated performance for this base-classifier will decrease (line 6). After this, base-classifier m_j can be trained with the example (line 7). When, at any time, a test example needs to be classified, the ensemble looks up the highest value of p_j and lets the corresponding ensemble member make the prediction.

The concept of an active classifier can also be extended to multiple active classifiers. Rather than selecting the best classifier on recent predictions, we can select the best k classifiers, whose votes for the specified class-label are all weighted according to some weighting schedule. First, we can weight them all equally. Indeed, when using this voting schedule and setting $k = |M|$, we would get the same behaviour as the Majority Vote Ensemble, as described by van Rijn et al. [127], which performed only averagely. Alternatively, we can use Online Performance Estimation to weight the

votes. This way, the best performing classifier obtains the highest weight, the second best performing classifier a bit less, and so on. Formally, for each $y \in Y$ (with Y being the set of all class labels):

$$votes_y = \sum_{m_j \in M} P(m_j, i, \alpha, L) \cdot B(l'_j(PS_i), y) \tag{5.4}$$

where M is the set of all models, l'_j is the labelling function produced by model m_j and B is an indicator function, returning 1 iff l'_j voted for class label y and 0 otherwise. Other functions regulating the voting process can also be incorporated, but are beyond the scope of this research. The label y that obtained the highest value $votes_y$ is then predicted.

BLAST has been implemented in the MOA framework, and can be obtained in the appropriate OpenML MOA package[1].

5.3.4 Meta-Feature Ensemble

Alternatively, we can use meta-learning to dynamically select between various classifiers. Several notable techniques for heterogeneous model combination in the data stream setting rely on meta-learning [125, 126, 133]. These techniques divide the data stream in windows of size w. Over each of these windows, a set of meta-features is calculated, and this is repeated for all previously known data streams. Using these meta-features, a meta-classifier is trained to predict the best classifier for the *next* window given the meta-features of the current window. Any batch classifier can be used as the meta-classifier. Usually, Random Forest are used, since these have proven to work well in the context of algorithm selection [144].

Table 5.1 shows the meta-features per category that are used in this work. The first four categories are the commonly used 'SSIL' features. The 'Drift detection' features were first introduced in [125]. These are obtained by running a drift detector (Adwin or DDM) on the dataset, and measure the number of changes or warnings in a given interval. The motivation behind this is that for volatile streams, where many changes are detected, simple classifiers might be preferred, whereas in steady streams, more complex classifiers have the opportunity to fit a good model.

The Stream Landmarkers represent the output of Online Performance Estimation as meta-feature. After each window, Eq. 5.1 is used to give an estimation on how well each base-classifier performed on that window. These performance estimations are then used as meta-features. In fact, the Windowed version of BLAST has the same information, when the grace period is equal to the window size.

[1]Available on Maven Central: `http://search.maven.org/` (artifact id: openmlmoa)

Table 5.1: Meta-features used by the Meta-Feature Ensemble.

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, Kappa and Area under the ROC Curve of the following classifiers: Decision Stump, J48 (confidence factor: 0.01), k-NN, NaiveBayes, REP Tree (maximum depth: 3)
Drift detection [125]	Changes by Adwin (Hoeffding Tree), Warnings by Adwin (Hoeffding Tree), Changes by DDM (Hoeffding Tree), Warnings by DDM (Hoeffding Tree), Changes by Adwin (Naive Bayes), Warnings by Adwin (Naive Bayes), Changes by DDM (Naive Bayes), Warnings by DDM (Naive Bayes)
Stream Landmarkers	Accuracy Naive Bayes on previous window, Accuracy k-NN on previous window, ...

5.4 Experimental Setup

In order to establish the utility of Online Performance Estimation and the two proposed techniques, we conduct experiments using a large set of data streams. The data streams and results of all experiments are made publicly available in OpenML for the purposes of verifiability, reproducibility and generalizability.

5.4.1 Data Streams

The data streams are a combination of real world data streams and synthetically generated data commonly used in data stream research (e.g., [10, 13, 125]). Many contain a natural drift of concept. Opposed to batch classification, in data stream classification it is quite natural to test techniques on synthetically generated data, as real world data is hard to obtain. We estimate the performance of the methods using the prequential method: each observation is used as a test example first and as a training example afterwards [57]. In total, a set of 60 different data streams is used.

Table 5.2: Data streams used for the Data Stream experiment.

name	obs.	atts.	cls.
SEA(50)	1,000,000	4	2
SEA(50000)	1,000,000	4	2
Stagger1	1,000,000	4	2
Stagger2	1,000,000	4	2
Stagger3	1,000,000	4	2
airlines	539,383	8	2
codrnaNorm	488,565	9	2
electricity	45,312	9	2
Agrawal1	1,000,000	10	2
BNG(tic-tac-toe)	39,366	10	2
BNG(cmc)	55,296	10	3
BNG(page-blocks)	295,245	11	5
Hyperplane(10;0001)	1,000,000	11	5
Hyperplane(10;001)	1,000,000	11	5
RandomRBF(0;0)	1,000,000	11	5
RandomRBF(10;0001)	1,000,000	11	5
RandomRBF(10;001)	1,000,000	11	5
RandomRBF(50;0001)	1,000,000	11	5
RandomRBF(50;001)	1,000,000	11	5
pokerhand	829,201	11	10
BNG(solar-flare)	1,000,000	13	3
BNG(bridges_version1)	1,000,000	13	6
BNG(heart-statlog)	1,000,000	14	2
BNG(wine)	1,000,000	14	3
BNG(heart-c)	1,000,000	14	5
BNG(vowel)	1,000,000	14	11
adult	48,842	15	2
BNG(JapaneseVowels)	1,000,000	15	9
BNG(credit-a)	1,000,000	16	2
BNG(labor)	1,000,000	17	2

name	obs.	atts.	cls.
BNG(vote)	131,072	17	2
BNG(pendigits)	1,000,000	17	10
BNG(letter)	1,000,000	17	26
BNG(zoo)	1,000,000	18	7
BNG(lymph)	1,000,000	19	4
BNG(vehicle)	1,000,000	19	4
BNG(hepatitis)	1,000,000	20	2
BNG(segment)	1,000,000	20	7
BNG(credit-g)	1,000,000	21	2
BNG(mushroom)	1,000,000	23	2
BNG(SPECT)	1,000,000	23	2
LED(50000)	1,000,000	25	10
AirlinesCodrnaAdult	1,076,790	30	2
BNG(trains)	1,000,000	33	2
BNG(ionosphere)	1,000,000	35	2
BNG(dermatology)	1,000,000	35	6
BNG(soybean)	1,000,000	36	19
BNG(kr-vs-kp)	1,000,000	37	2
BNG(satimage)	1,000,000	37	6
BNG(anneal)	1,000,000	39	6
BNG(waveform-5000)	1,000,000	41	3
covertype	581,012	55	7
BNG(spambase)	1,000,000	58	2
BNG(sonar)	1,000,000	61	2
BNG(optdigits)	1,000,000	65	10
CovPokElec	1,455,525	73	10
BNG(mfeat-fourier)	1,000,000	77	10
vehicleNorm	98,528	101	2
20_newsgroups.drift	399,940	1,001	2
IMDB.drama	120,919	1,002	2

Table 5.2 shows all datasets used in this experiment. Some of the used data streams and data generators are described below.

SEA Concepts The SEA Concepts Generator [143] generates three numeric attributes from a given distribution, of which only the first two are relevant. The class that needs to be predicted is whether these values exceed a certain threshold. Several SEA Concept generated data streams based on different data distributions can be joined together in order to simulate concept drift.

STAGGER The STAGGER Concepts Generator [137] generates descriptions of geometrical objects. Each instance describes the size, shape and colour of such an object. A STAGGER concept is a binary classification rule distinguishing between the two classes, e.g., all blue rectangles belong to the positive class.

Rotating Hyperplanes The Rotating Hyperplane Generator [73] generates a high-dimensional hyperplane. Instances represent a point in this high-dimensional space. The task is to predict whether such a point is below the hyperplane. Concept drift can be introduced by rotating and moving the hyperplane.

LED The LED Generator [25] generates instances based on a LED display. Attributes represent the various LED lights, and the task is to predict which digit is represented. In order to add noise, attributes can display the wrong value with a certain probability. Furthermore, additional (irrelevant) attributes can be added.

Random RBF The Random RBF Generator generates a number of centroids. Each has a random position in Euclidean space, standard deviation, weight and class label. Each example is defined by its coordinates in Euclidean Space and a class label referring to a centroid close by. Centroids move at a certain speed, generating gradual concept drift.

Bayesian Network The Bayesian Network Generator [125] takes a batch data set as input, builds a Bayesian Network over it, and generates instances based on the probability tables. As the Bayesian Network does not change over time, it is unlikely that a native form of concept drift occurs in the data stream.

Composed Data Streams Common batch datasets can be appended to each other, forming one combined dataset covering all observations and attributes, containing small periods or abrupt concept drift. This is commonly done with the 'Covertype', 'Pokerhand' and 'Electricity' dataset [14, 118]. We applied a similar merge to the 'Airlines', 'CodeRNA' and 'Adult' dataset, forming 'AirlinesCodernaAdult'. The original datasets are normalized before this operation is applied.

IMDB.drama The 'IMDB' dataset contains $120{,}919$ movie plots. Each movie is represented by a bag of words of the globally $1{,}000$ most occurring words. Originally, it is a multi-label dataset, with binary attributes indicating whether a movie belongs to a given genre. We predict whether it is in the drama genre, which is the most frequently occurring [118].

20 **Newsgroups** The original 20 Newsgroup dataset contains $19{,}300$ newsgroup messages, each represented as a bag of words of the $1{,}000$ most occurring words. Each instance is part of at least one newsgroup. This data set is commonly converted into

Table 5.3: Classifiers used in the experiments. All as implemented in MOA 2016.04 by Bifet et al. [13], default parameter settings are used unless stated otherwise.

Classifier	Model type	Parameters
Naive Bayes	Bayesian	
Stochastic Gradient Descent	SVM	Loss function: Hinge
k Nearest Neighbour	Lazy	$k = 10$, $w = 1{,}000$
Hoeffding Option Tree	Option Tree	
Perceptron	Neural Network	
Random Hoeffding Tree	Decision Tree	
Rule Classifier	Decision Rules	

20 binary classification problems, with the task to determine whether an instance belongs to a given newsgroup. We append these data sets to each other, resulting in one large binary-class dataset containing $386{,}000$ records with 19 shifts in concept [118].

5.4.2 Parameter Settings

The heterogeneous ensembles contain the seven classifiers from Table 5.3 as base-classifiers. The table also states the model type, as described in Chapter 2. The classifiers are selected using the dendrogram of Figure 4.14 (page 69), loosely omitting some simple models such as No Change, Majority Class and Decision Stumps. BLAST is tested with both Fading Factors ($\alpha = 0.999$) and the Windowed approach ($w = 1{,}000$). For both versions $k = 1$, thus a single best classifier is chosen for every prediction. We explore the effect of larger values for k in Chapter 5.5.2. The Meta-Feature Ensemble also uses windows of size $1{,}000$.

5.4.3 Baselines

We compare the results of the defined methods with the *Best Single Classifier*. Each heterogeneous ensemble consists of n base-classifiers. The one that performs best on average over all data streams is considered the best single classifier. Following Figure 5.5, the Hoeffding Option Tree is the best classifier (this is also confirmed by [127]). This baseline enables us to measure the potential accuracy gains of adding more classifiers (at the cost of using more computational resources).

Furthermore, we compare against the *Majority Vote Ensemble* (same base-classifiers as in Table 5.3), which is a heterogeneous ensemble that predicts the label that is predicted by most ensemble members. This enables us to measure the potential accuracy

gain of using Online Performance Estimation over just naively counting the votes of base-classifiers. Finally, we also compare the techniques to state of the art homogeneous ensembles, such as Online Bagging, Leveraging Bagging, and Accuracy Weighted Ensemble. In order to understand these a bit better, we provide some results.

Figure 5.6 shows violin plots of the performance of Accuracy Weighted Ensemble (left bars, red), Leveraging Bagging (middle bars, green) and Online Bagging (right bars, blue), with an increasing number of ensemble members. Accuracy Weighted Ensebmble (AWE) uses J48 trees as ensemble members, both Bagging schemas use Hoeffding Trees. Naturally, as the number of members increases, both accuracy and run time increase, however Leveraging Bagging performs eminently better than the others. Leveraging Bagging using 16 ensemble members already outperforms both AWE and Online Bagging using 128 ensemble members, based on median accuracy. This performance also comes at a cost, as it uses considerably more run time than both other techniques, even when containing the same number of members. Acurracy Weighted Ensemble performs pretty constant, regardless of the number of ensemble members. As the ensemble size grows, both accuracy and run time slightly increase. We will compare BLAST against the homogeneous ensembles consisting of 128 members.

5.5 Results

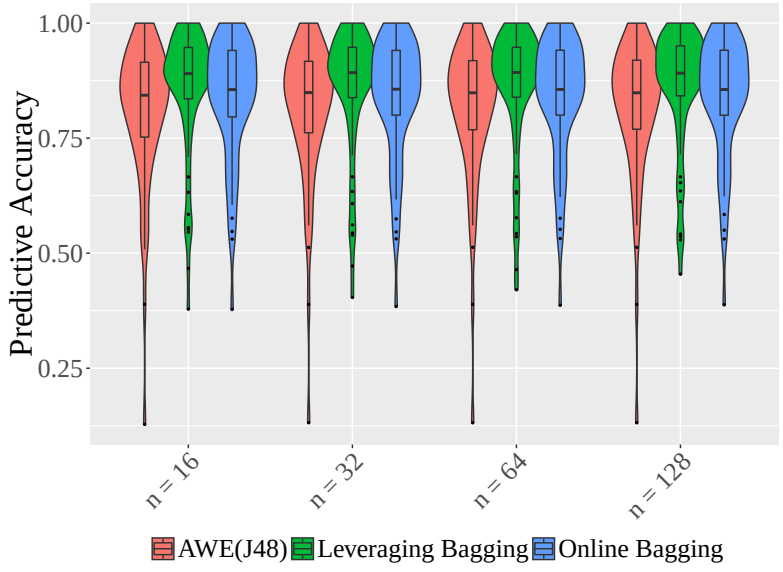
In this chapter we present and discuss the experimental results.

5.5.1 Ensemble Performance

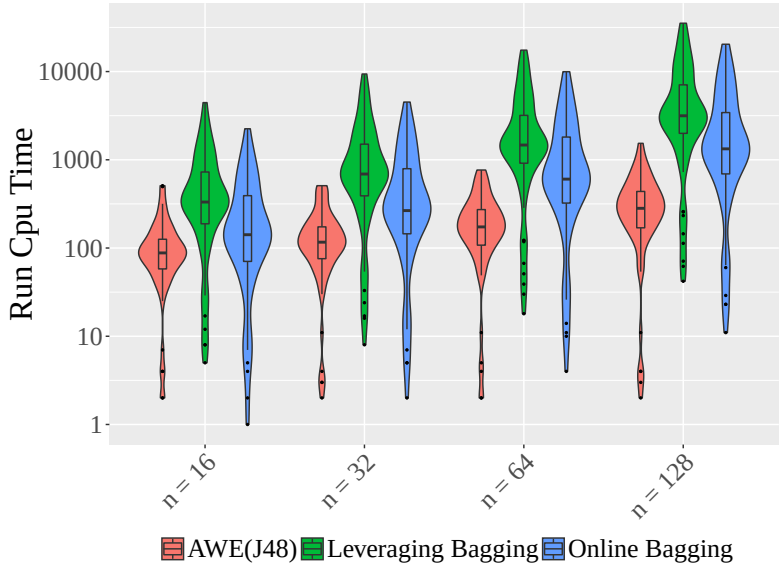
We run all techniques on the set of 60 data streams. The results are shown in Figure 5.7. As the Meta-Feature ensemble is a virtual classifier, there are no run time results available. However, as it builds the same set of base-classifiers as the Majority Vote Ensemble and both versions of BLAST, it is plausible that the run time is in the same order of magnitude.

An important observation is that both versions of BLAST are competitive with state of the art ensembles. These results suggest that Online Performance Estimation is a useful technique when applied to data stream ensembles. Also note that BLAST requires far fewer run time than the ensemble techniques. A peculiar observation is that BLAST also outperforms the Meta-Feature Ensemble, which has access to the same information and thus is supposed to perform at least equally good.

Figure 5.8 shows the accuracy of four heterogeneous ensemble techniques per data stream. In order to not overload the figure, we only show BLAST (with fading

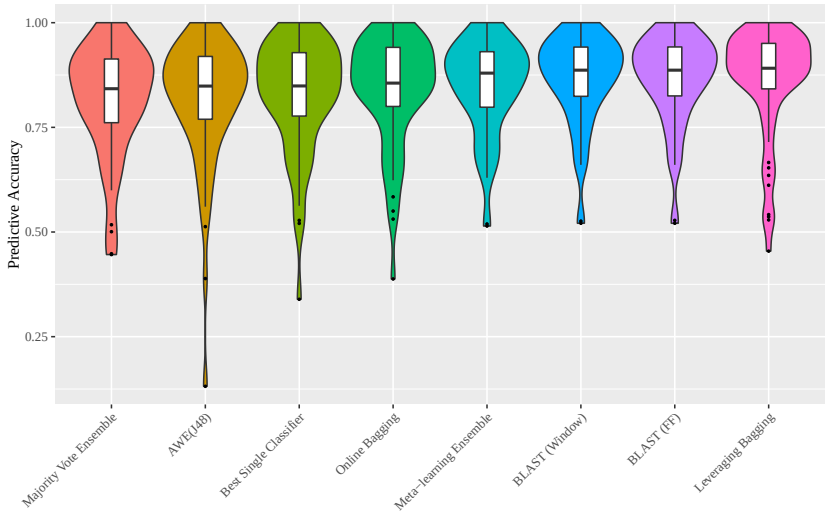


(a) Accuracy

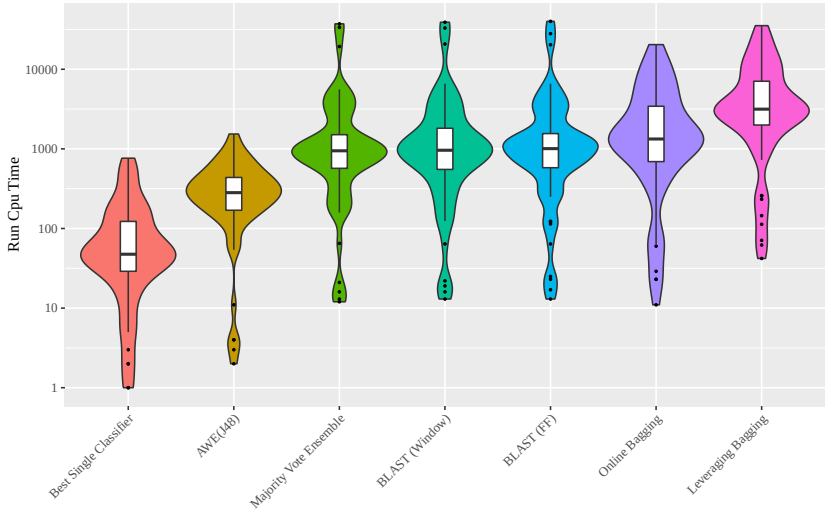


(b) Run time

Figure 5.6: Effect of the ensemble size parameter.



(a) Accuracy



(b) Run time

Figure 5.7: Performance of the various meta-learning techniques averaged over 60 data streams.

factors), the Meta-Feature Ensemble, Leveraging Bagging and the Best Single Classifier.

All techniques outperform the Best Single Classifier consistently. Especially on data streams where the performance of the Best Single Classifier is mediocre (Figure 5.8 bottom part), accuracy gains are eminent. The difference between Leveraging Bagging and BLAST is harder to assess. Although Leveraging Bagging seems to be slightly better in many cases, there are some clear cases where there is a big difference in favour of BLAST (see, e.g., the difference in data stream 'IMDB.drama'). Despite Leveraging Bagging is best on most data streams, other techniques are better on average.

To assess statistical significance of the results, we use the Friedman test with post-hoc Nemenyi test to establish the statistical relevance of our results. These tests are considered the state of the art 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.

The results of the Nemenyi test are shown in Figure 5.9. It plots the average rank of all methods and the critical difference. Classifiers for which not a statistically significant difference was found are connected by a black line. For all other cases, there was a significant difference in performance, in favour of the classifier with the better (lower) average rank. We performed the test based on Accuracy and run time. Again, no run time results are recorded for the Meta-Feature Ensemble.

Figure 5.9a shows that there is no statistically significant difference in terms of accuracy between BLAST (using Fading Factors) and two of the homogeneous ensembles (i.e., Leveraging Bagging and Online Bagging using 128 Hoeffding Trees). BLAST (Window) does perform significantly worse than Leveraging Bagging. Meta-Feature Ensemble is significantly worse than Leveraging Bagging and BLAST (with Fading Factors). As expected, the Best Single Classifier and the Majority Vote Ensemble perform significantly worse than most other techniques. Clearly, combining heterogeneous ensemble members by simply counting votes does not work in this setup. It seems that poor results from some ensemble members outweigh the diversity.

When comparing the techniques on computational efficiency, the overall outcome is different, as reflected by Figure 5.9b. The best single classifier (Hoeffding Option Tree) requires fewest resources. There is no significant difference in resources between BLAST (FF), BLAST (Window), Majority Vote Ensemble and Online Bagging. This makes sense, as these the first three operate on the same set of base-classifiers. The Accuracy Weighted Ensemble performs really well in terms of run time, even though it uses 128 ensemble members. Due to its efficient trainings technique, adding more classifiers does not necessarily result in more run time. Altogether, BLAST (FF) performs equivalent to both Bagging schemas in terms of accuracy, while using signi-

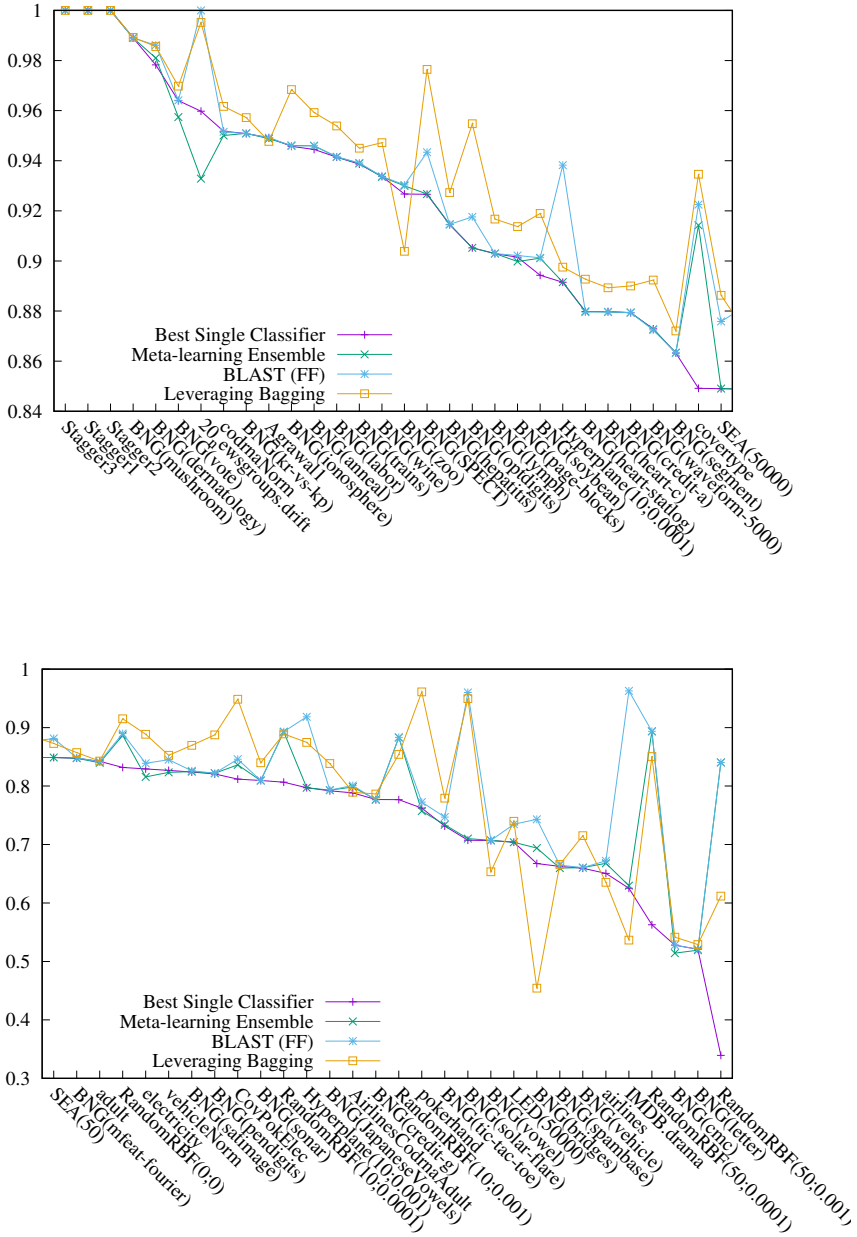
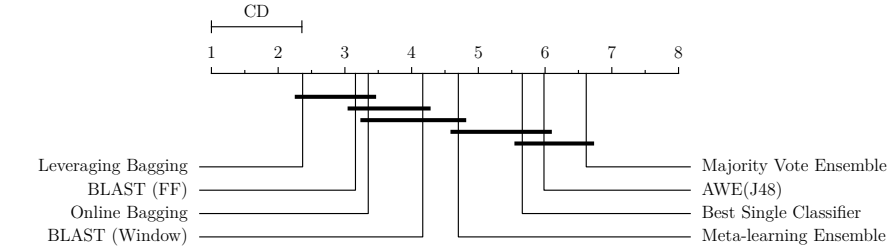
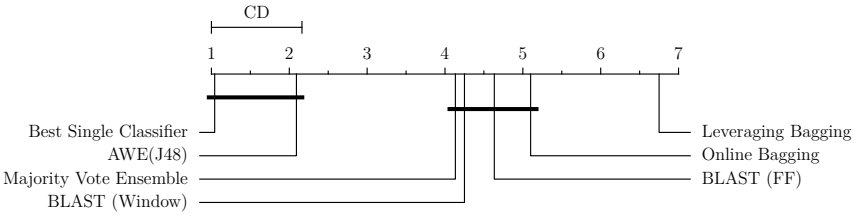


Figure 5.8: Accuracy per data stream, sorted by accuracy of the best single classifier.



(a) Accuracy



(b) Run time

Figure 5.9: Results of Nemenyi test. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

ficantly fewer resources.

5.5.2 Effect of Parameters

In this chapter we study the effect of the user-defined parameters of BLAST.

5.5.2.1 Window size and decay rate

First, for both version of BLAST, there is a parameter that controls the rate of dismissal of old observations. For BLAST (FF) this is the α parameter (the fading factor); for BLAST (Windowed) this is the w parameter (the window size). The α parameter is always in the range $[0, 1]$, and has no effect on the use of resources. The window parameter can be in the range $[0, n]$, where n is the size of the data stream. Setting this value higher results in bigger memory requirements, although these are typically neglectable compared to the memory usage of the base-classifiers.

Figure 5.10 shows violin plots and box plots of the effect of varying these parameters. The effect of the α (a) value on BLAST (FF) is displayed in the left (red)

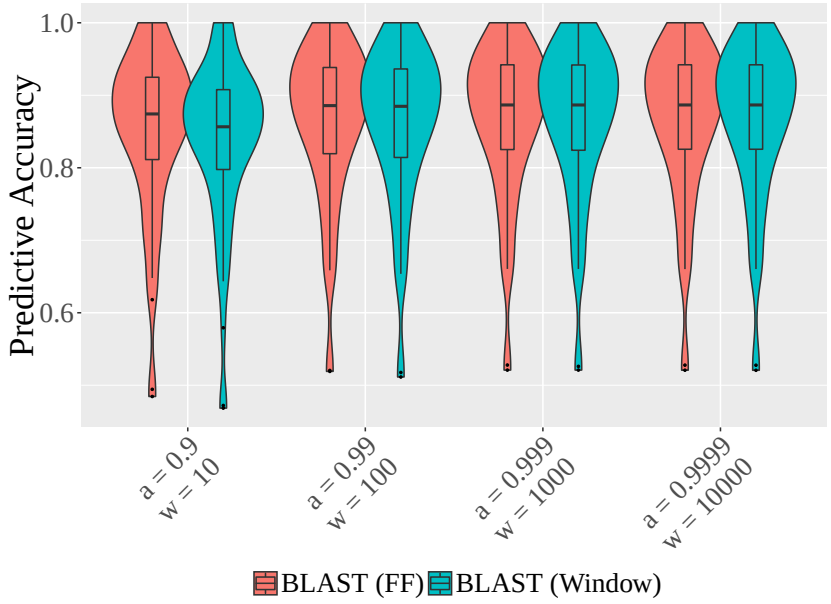


Figure 5.10: Effect of the decay rate and window parameter.

violin plots; the effect of the window (w) value on BLAST (Window) is displayed in the right (blue) violin plots. Even though it is hard to draw general conclusions from this, the trend over 60 data streams seems to be that setting this parameter too low results in suboptimal accuracy. Arguably, this is good in highly volatile streams when concepts change rapidly, but in general we do not want to dismiss old information too quickly. Altogether, BLAST (FF) seems to be slightly more robust, as the investigated values of the α parameter does not perceptibly influence the performance.

5.5.2.2 Grace parameter

Prior work by Rossi et al. [133] and van Rijn et al. [127] introduced a grace parameter that controls the number of observations for which the active classifier was not changed. This potentially saves resources, especially when the algorithm selection process involves time consuming operations such as the calculation of meta-features. On the other hand, it can be seen that in a data stream setting where concept drift occurs, in terms of performance it is always optimal to act on changed behaviour as fast as possible. Although we have omitted this parameter from the formal definition of BLAST in Chapter 5.3, similar behaviour can be obtained by updating the active

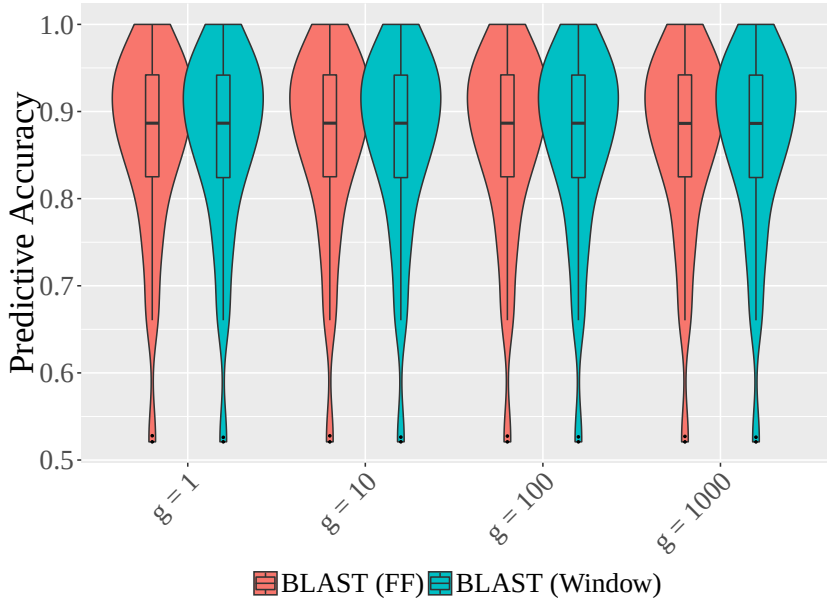


Figure 5.11: Effect of the grace parameter on accuracy. The x-axis denotes the grace, the y-axes the performance. BLAST (Window) was ran with $w = 1{,}000$; BLAST (FF) was ran with $\alpha = 0.999$.

classifier only at certain invervals. Formally, a grace period can be enforced by only executing Eq. 5.3 when $c \mod g = 0$, where (following earlier definitions) c is the index of the current observation, and g is a grace period defined by the user.

Figure 5.11 shows the effect of the (hypothetical) grace parameter on performance, averaged over 60 data streams. The plots do not indicate that there is much difference in performance. Moreover, the algorithm selection phase of BLAST simply depends on finding the maximum element in an array. Therefore, the grace period would not have any influence on the required resources.

5.5.2.3 Number of active classifiers

Rather than selecting one active classifier, multiple active classifiers can be selected that all vote for a class label. The votes of these classifiers can either contribute equally to the final vote, or be weighted accouring to their estimated performance. We used BLAST (FF) to explore the effect of this parameter. We varied the number of active classifiers k from one to five, and measured the performace according to both voting

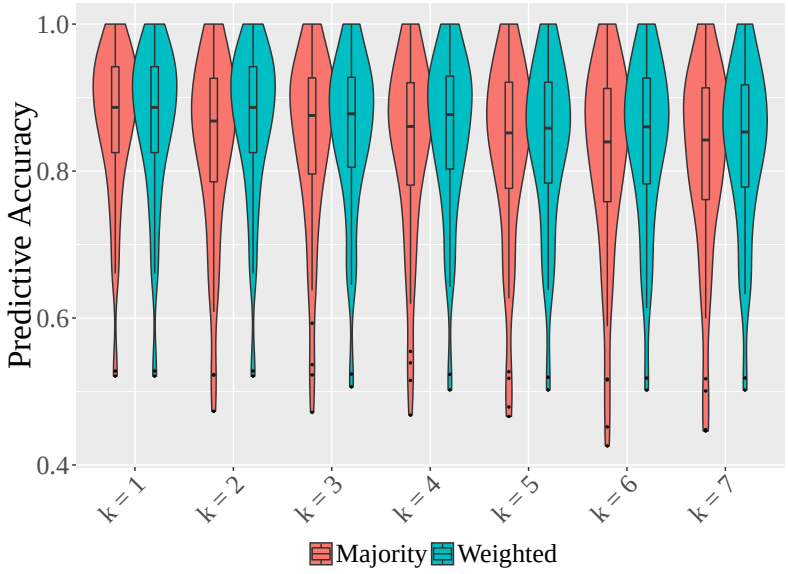
schemas. Figure 5.12 shows the results.

Figure 5.12a shows how the various strategies perform when evaluated using predictive accuracy. We can make several observations to verify the correctness of the results. First, the results of both strategies are equal when $k = 1$, as the algorithm selects only one classifier, weights are obsolete. Second, the result of the Majority weighting schema for $k = 7$ is equal to the score of the Majority Weight Ensemble (see Figure 5.7a), which is also correct, as these are the same by definition. Finally, when using the weighted strategy, setting $k = 2$ yields exactly the same scores for accuracy as setting $k = 1$. This also makes sense, as it is guaranteed that the second best base-classifier always has a lower weight as the best base-classifier, and thus it is incapable of changing any prediction.

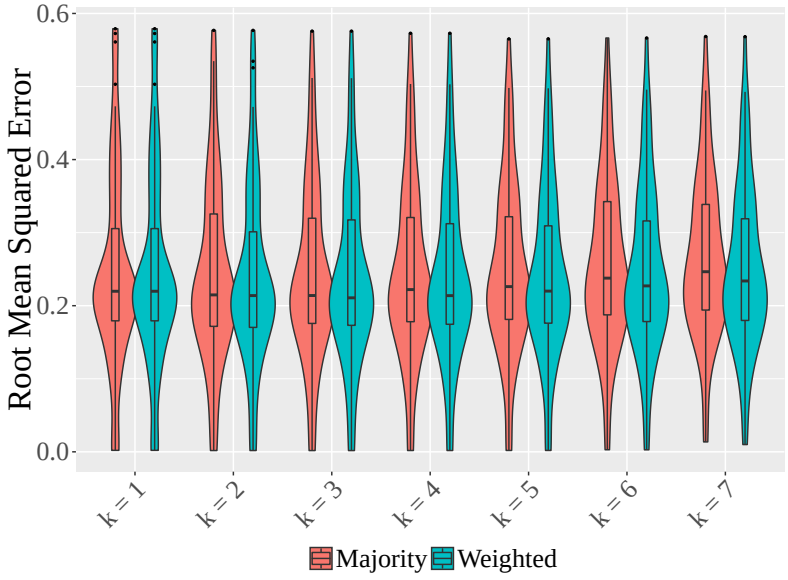
In all, it seems that increasing the number of active classifiers is not beneficial for accuracy. Note that this is different from adding more classifiers in general, which clearly would not decrease performance results. This behaviour is different from the classical approach, where adding more classifiers (which inherently are all active) yield better results up to a certain point [30]. However, in the data stream setting we deal with a time component, and we can actually measure which classifiers performed well on recent intervals. By increasing the number of active classifiers, we would add classifiers to the vote of which we have empirical evidence that they performed worse on recent observations.

Similarly, Figure 5.12b shows the Root Mean Squared Error (RMSE). RSME is typically used as an evaluation measure for regression, but can be used in classification problems to assess the quality of class confidences. For every prediction, the error is considered to be the difference between the class confidence for the correct label and 1. This means that if the classifier had a confidence close to 1 for the particular class label, a low error is recorded, and vice versa. The violin plots indicate that adding more active classifiers can lead to a lower median error. This also makes sense, as the Root Mean Squared Error tends to punish classification errors harder when these are made with a high confidence. We observe this effect until $k = 3$, after which adding more active classifiers starts to lead to a higher RMSE. It is unclear why this effect holds until this particular value.

All together, from this experiment we conclude that adding more active classifiers in the context of Online Performance Estimation does not necessarily yields better results beyond selecting the expected best classifier at that point in the stream. This might be different when using more base-classifiers, as we would expect to have more similarly performing classifiers on each interval. As we expect to measure this effect when using orders of magnitude more classifiers, this is considered future work. Clearly, when using multiple active classifiers, weighting their votes using online performance estimation seems beneficial.

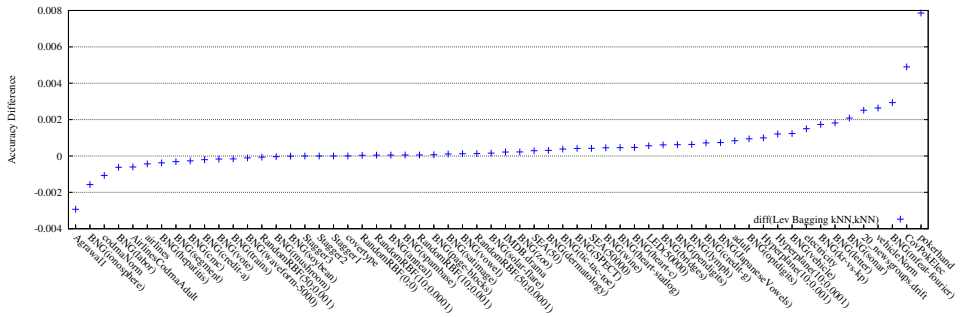


(a) Accuracy

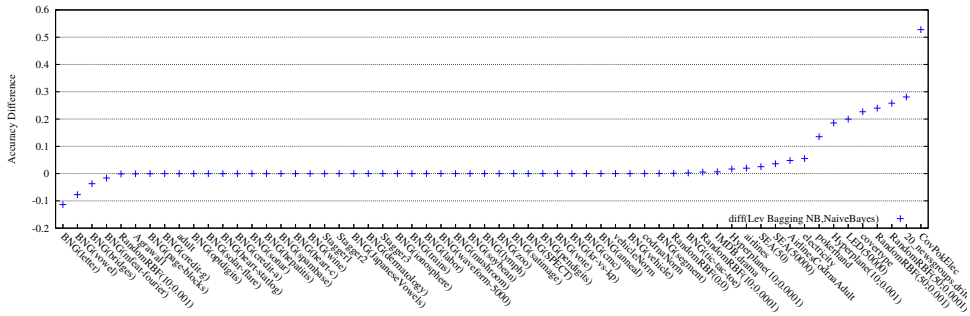


(b) Root Mean Squared Error

Figure 5.12: Performance for various values of k, weighted votes versus unweighted votes.



(a) Leveraged Bagging k-NN Vs. k-NN



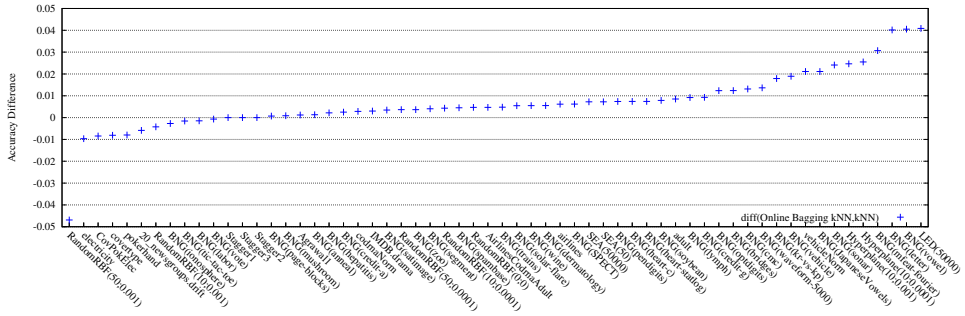
(b) Leveraged Bagging Naive Bayes Vs. Naive Bayes

Figure 5.13: Performance differences between Leveraging Bagging ensembles and single classifiers.

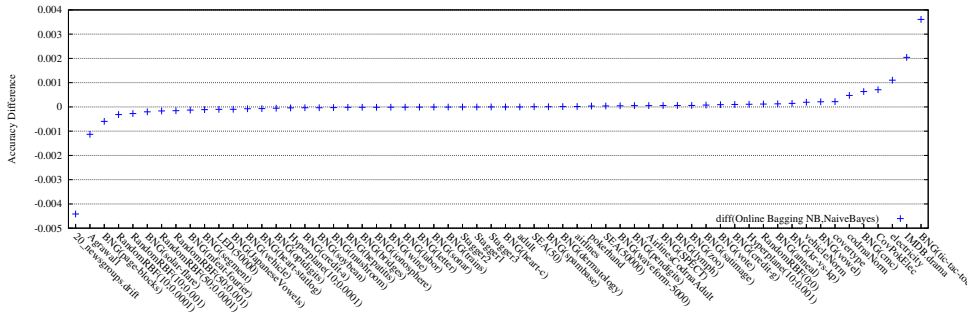
5.6 Designed Serendipity

The philosophy of openly sharing results and working on massive amounts of data streams leads to unexpected results that would possibly remain undiscovered otherwise. In prior work was already observed that applying a Bagging technique like Leveraged Bagging improves classification performance for k Nearest Neighbour [125]. This is a counter-intuitive result, as Breiman [23] already conjectured that Bagging will not affect stable classifiers in the batch setting; k Nearest Neighbour with $k = 10$ is considered a stable classifier. A follow-up study has been conducted for the data stream setting, covering both k Nearest Neighbour and Naive Bayes, and comparing the performance over two Bagging Schema's, i.e., Online Bagging and Leveraging Bagging [129].

Figure 5.13 and Figure 5.14 show the results of the experiments. The x-axis shows



(a) Online Bagging k-NN Vs. k-NN



(b) Online Bagging Naive Bayes Vs. Naive Bayes

Figure 5.14: Performance differences between Online Bagging ensembles and single classifiers.

the dataset, the y-axis shows the difference in performance between the bagging schema and the single classifier. Note that the single classifier performed better on datasets where the difference is below zero, and vice versa for datasets where the difference is above zero. Datasets are ordered by this difference.

Figure 5.13a shows a similar trend as seen in the figure presented in [125]. There are few data streams on which k Nearest Neighbour performs best, but many data streams on which Leveraging Bagging k Nearest Neighbour performs best. One notable observation is the difference in scale between Figure 5.13b describing the effect of using Naive Bayes in a Leveraging Bagging schema, and the other figures. In most cases, the effect of a Bagging schema can attribute to performance gains of few percentages. However in the case of Leveraged Bagging Naive Bayes, the performance gain can lead up to 50% in the 'CovPokElec' dataset, but also other data sets show eminent improvements.

Among the 15 data streams on which Leveraged Bagging improves upon Naive Bayes the most, many presumably contain concept drift. Compared to k Nearest Neighbour, Naive Bayes' performance is quite poor on these data streams[2]. Apparently, k Nearest Neighbour's natural protection against concept drift (it removes old instances as new ones come in) makes it perform quite well. Note that Leveraging Bagging is a Bagging technique combined with a change detector and leveraging more randomness. When using Naive Bayes in a Leveraging Bagging schema, the change detector ensures that Naive Bayes also obtains this performance increase.

Figure 5.14b shows what we would expect to see when applying Online Bagging, which is a pure form of Bagging described in Chapter 3, to a stable classifier. The differences in accuracy are small and the performance gains are equally divided between the single classifier and the bagging schema.

From this we learn three things.

- First, when applied in a normal Bagging schema, Naive Bayes classifiers exhibit similar behaviour as in the batch setting (Figure 5.14b).
- Second, this result does not hold for Leveraging Bagging, which also embodies a change detector. This can have a big influence on classification results (Figure 5.13b).
- Finally, concept drift has a big influence on the applicability of bagging schemas.

5.7 Conclusions

This chapter covered Data Streams and various Machine Learning techniques that operate on them. We surveyed the Online Performance Estimation framework, which can be used in data stream ensembles to weight the votes of ensemble members, in particular when using fundamentally different model types. Online Performance Estimation measures the performance of all base-classifiers on recent training examples. Using two different performance estimation functions, it can be used to built heterogeneous ensembles.

BLAST is a heterogeneous ensemble technique based on Online Performance Estimation that selects the single best classifier on recent predictions to classify new observations. We have integrated both performance estimation functions into BLAST. The Meta-Feature Ensemble uses a variety of traditional meta-features and meta-features obtained from the Online Performance Estimation framework. The introduced techniques were developed using experimental results stored in OpenML. Em-

[2]This information can not be deduced from the figures, therefore the reader is referred to OpenML or Table 3 of [129]

pirical evaluation shows that BLAST with fading factors outperforms the other methods. This is most likely because Fading Factors are better able to capture typical data stream properties, such as changes of concepts. When this occurs, there will also be changes in the performances of ensemble members, and the fading factors adapt to this relatively fast. Based on an empirical evaluation covering 60 data streams, we observe that BLAST is statistically equivalent to current state of the art ensembles while using significantly fewer resources.

We also evaluated the effect of the method's parameters on the performance. The most important parameter proves to be the one controlling the performance estimation function: α for the fading factor, controlling the decay rate, and w for the windowed approach, determining the window size. Our results show that the optimal value for these parameters is dependent on the given dataset, although setting this too low turns out to have a worse effect on accuracy than setting it too high.

To select the classifiers included in the heterogeneous ensemble, we used the hierarchical clustering of 25 commonly used data stream classifiers that was presented in Figure 4.14 (page 69). We used this clustering to gain methodological justification for which classifiers to use, although the clustering is mainly a guideline. A human expert can still determine to deviate from the resulting set of algorithms, in order to save resources. The resulting dendrogram also has scientific value in itself. It confirms some well-established assumptions regarding the typically used classifier taxonomy in data streams, that have never been tested before. Many of the classifiers that were suspected to be similar were also clustered together, for example the various decision trees, support vector machines and gradient descent models all formed their own clusters. Moreover, some interesting observations were made that can be investigated in future work. For instance, the Rule Classifier used turns out to perform averagely, and was rather far removed from the decision trees, whereas we would expect it to perform better and be clustered closer to the decision trees.

Utilizing the Online Performance Estimation framework opens up a whole new line of data stream research. Rather than creating more data stream classifiers, combining them in a suitable way can elegantly lead to highly improved results that effortlessly adapt to changes in the data stream. More than in the classical batch setting, memory and time are of crucial importance. Experiments suggest that the selected set of base-classifiers has a substantial influence on the performance of the ensemble. Research should be conducted to explore what model types best complement each other, and which work well together given a constraint on resources. Combining data stream research with the open approach of OpenML led to new knowledge and techniques. We believe that by exploring these possibilities we can further push the state of the art in data stream ensembles.

6 Combining Accuracy and Run Time

Meta-learning focuses on finding classifiers and parameter settings that work well on a given dataset. Evaluating all possible combinations typically takes too much time, hence many solutions have been proposed that attempt to predict which classifiers are most promising to try. As discussed in Chapter 3, the first recommended classifier is not always the correct choice, so multiple recommendations should be made, making this a ranking problem rather than a classification problem.

Even though this is a well studied problem, in the meta-learning literature there is no common way of evaluating these. We advocate the use of Loss Time Curves, as used in the field of optimization. These visualize the amount of budget (time) needed to converge to an acceptable solution. We investigate two methods that utilize the measured performances of classifiers on small samples of data to make such recommendation, and adapt it so that these works well in Loss Time space. Experimental results show that this method converges extremely fast to an acceptable solution.

OpenML was used as an experiment repository. The datasets and tasks that were used take many resources (time and memory) to model, so this work was greatly accelerated by including prior results that were collaboratively generated.

6.1 Introduction

When presented with a new classification problem, a key challenge is to identify a classifier and parameter settings that obtain good predictive performance. This problem is known as the *Algorithm Selection Problem* [119]. Since many classifiers exist, all containing a number of parameters that potentially influence predictive performance, this is a challenging problem. Performing a cross-validation evaluation procedure on

all possible combinations of classifiers and parameters (e.g., using a grid search) is typically infeasible and suboptimal, for the reasons mentioned in Chapter 3. The field of *meta-learning* attempts to solve this by learning from prior examples. Typically, a set of classifiers is recommended based on the performance on similar datasets.

The meta-learning method SAM [87] identifies similar datasets based on the *learning curves* of classifiers trained on them, and recommends the classifier that performs best on these similar datasets. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size [113]. Although the results are convincing, it does not take into account some important aspects of algorithm selection. First, it only recommends the single best classifier, rather than a ranking of candidates. Second, it does not take the training time of the models into account, making it unable to distinguish between fast and slow classifiers. Indeed, in practical applications there is typically a budget (e.g., limited time or a maximum number of cross-validation runs) within which a number of classifiers can be evaluated. As such, the meta-learning method should be evaluated on how well it performs within a given budget.

Another popular meta-learning technique is *Active Testing* [88]. It recommends a ranking of classifiers, advising in which order these should be cross-validated to check their applicability to the inspected dataset. This ranking is dynamically updated, based on results from earlier performed cross-validation tests on the dataset at hand. This method also does not take the training time of the models into account, making it unable to distinguish between fast and slow classifiers.

This chapter covers the following contributions. We extend the aforementioned techniques so that these produce a ranking of classifiers and takes into account the run times of classifiers. For this, a new evaluation measure is explored, $A3R'$, capable of trading of accuracy and run time. Furthermore, we study the performance of this method in Loss space and Loss Time space. We will argue that Loss Curves as presented in [88] are biased, and propose the use of Loss Time Curves, as commonly used in Optimization literature (e.g., [74]). Finally, we compare the method against a range of alternative methods, including a rather strong baseline that recommends the classifier that performed best on a small sample of the data [52]. Our proposed technique dominates the baseline methods in some scenarios. Moreover, our results suggest that a simple, sample-based baseline technique has been mistakenly neglected in the literature. Finally, we will see that meta-learning techniques that adopt $A3R'$ improve their performance in Loss Time space.

This chapter is organized as follows. Chapter 6.2 surveys related work. Chapter 6.3 formalizes both methods, and adapt them to work in Loss Time space. Chapter 6.4 contains experiments. Chapter 6.5 concludes.

6.2 Related Work

Meta-learning aims to learn which learning techniques work well on what data [141]. A common task, known as the Algorithm Selection Problem [119], is to determine which classifier performs best on a given dataset. We can predict this by training a meta-model on meta-data comprised of dataset characterizations, i.e., *meta-features* [20], and the performances of different classifiers on these datasets. The same meta-features can be computed on each new dataset and fed to the meta-model to predict which classifiers will perform well.

Hence, the Algorithm Selection Problem is reduced to a Machine Learning problem. Meta-features are often categorized as either simple (e.g., number of examples, number of attributes), statistical (e.g., mean standard deviation of attributes, mean skewness of attributes), information theoretic (e.g., class entropy, mean mutual information) or landmarkers [108] (performance evaluations of simple classifiers). Many meta-learning studies follow this approach [125, 133, 144, 158, 163].

As argued in Chapter 3, meta-feature-based approaches have some intrinsic limitations. First, it is hard to construct a meta-feature set that adequately characterizes the problem space [86]. Second, the most successful meta-features, landmarkers, can be computationally expensive, limiting the options [108]. Finally, because not all classifiers can model all datasets, or take prohibitively long to do so, the meta-dataset usually contains many missing values, complicating the classification task.

In order to overcome these problems, Leite and Brazdil [86, 87] identify similar datasets based on partial learning curves. In this particular method, a *partial learning curve* is computed, using small samples, to identify similar datasets and use performance information from those datasets to extrapolate the learning curve. As such, running classifiers on these samples is rather cheap. There is also a clear connection with multi-armed bandit strategies [89], where results on a small sample determine whether it is worthwhile to continue learning.

Alternatively, the Best on Sample method uses the performance estimates of classifiers on a small subset (sample) of the data, and recommends the classifiers which perform best on this sample, in descending order [107]. The smaller this sample is, the fewer time this method takes to execute. Prior work is inconclusive about its performance. The authors of [107] suggest that this technique should be used as a baseline method in meta-learning research. The authors of [52] show that this information is not useful as a landmarker. Indeed, it has been correctly observed that learning curves sometimes cross, i.e., one classifier can outperform another on a small data sample, but can be surpassed when trained on the whole dataset [86]. However, this happens less often as the sample size increases, making this method quite reliable when using the right sample size, as we will see in Chapter 6.4.

These three methods all aim to recommend an algorithm with high accuracy. However, in a setting where multiple classifiers will be tried sequentially and the budget is time, it might make sense to first try many fast algorithms, rather than a few slow ones. This can be done by selecting classifiers based on a trade-off between accuracy and run time. Brazdil et al. [21] proposes *adjusted ratio of ratios* (ARR), which is defined as:

$$ARR^{d_i}_{a_p,a_q} = \frac{\frac{SR^{d_i}_{a_p}}{SR^{d_i}_{a_q}}}{1 + AccD \cdot \log_2\left(\frac{T^{d_i}_{a_p}}{T^{d_i}_{a_q}}\right)} \tag{6.1}$$

where $SR^{d_i}_{a_p}$ and $SR^{d_i}_{a_q}$ are the predictive accuracy (success rate) of classifiers a_p and a_q (respectively) on dataset d_i. Likewise, $T^{d_i}_{a_p}$ and $T^{d_i}_{a_q}$ are the run times of classifiers a_p and a_q (respectively) on dataset d_i. Finally, $AccD$ is a parameter controlled by the user, influencing the relative importance of accuracy to run time.

As was pointed out by [1], there are some problems with this measure: it is not monotonic, and even approaches infinity at some point. Therefore, the measure $A3R$ was introduced:

$$A3R^{d_i}_{a_p,a_q} = \frac{\frac{SR^{d_i}_{a_p}}{SR^{d_i}_{a_q}}}{\sqrt[r]{T^{d_i}_{a_p}/T^{d_i}_{a_q}}} \tag{6.2}$$

where, similar to ARR, $SR^{d_i}_{a_p}$ and $SR^{d_i}_{a_q}$ are the predictive accuracy (success rate) of classifiers a_p and a_q (respectively) on dataset d_i. Likewise, $T^{d_i}_{a_p}$ and $T^{d_i}_{a_q}$ are the run times of classifiers a_p and a_q (respectively) on dataset d_i. Finally, r is a parameter controlled by the user, influencing the relative importance of accuracy versus run time.

Although both ARR and $A3R$ are suitable for finding fast classifiers, these have not been used as such before. Experimental evaluations have focused on recommending classifiers that work well on this criterion. This seems a bit arbitrary. Indeed, we are still interested in finding the algorithm with the highest accuracy, however we want to find it as fast as possible (i.e., with fewest number of cross-validation test or time). In this respect our approach differs from earlier meta-learning approaches by using this measure to build a ranking of classifiers that finds a reasonable classifier as fast as possible.

6.3 Methods

In this chapter we describe two methods that extend the work of [86, 87, 88] in several ways. We consider a set A of classifiers, a_m ($m = 1, 2, 3, \ldots, M$). We also consider a set D of datasets, d_n ($n = 1, 2, 3, \ldots, N$), on which we have information on the performance of the classifiers in A (d_{new} is not in D). The total amount of trainings instances available for dataset d_n is denoted as $|d_n|$. Let $P_{m,n,s}$ and $P'_{m,n,s}$ denote the performance of classifier a_m on dataset d_n, for a given evaluation measure (e.g., predictive accuracy), using a sample size of s. Furthermore, Ω denotes the size of a dataset given by the context. Hence, $P_{m,n,\Omega}$ (equals $P_{m,n,|d_n|}$) denotes the performance of classifier a_m on the full dataset d_n.

6.3.1 Pairwise Curve Comparison

Various methods that are successful at recommending algorithms make use of so-called learning curves. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size [113]. The method proposed by [86] builds a partial learning curve for a pair of algorithms, and finds among earlier seen datasets (on which these algorithms were also run) the one that has the most similar partial learning curves. Based on the performance of the algorithms on that full dataset, it makes a recommendation. This idea is extended to multiple classifiers (rather than a pair) in [87].

In this chapter, we propose a novel method that extends the method as defined by [87] in three ways. First, it recommends a ranking of classifiers, rather than just a single best classifier. Second, it can take arbitrary evaluation measures into account, such as run time. Lastly, we introduce an optimization called *Smaller Sample*, that improves performance when the sizes of datasets differ a lot.

Let S be the set of samples of dataset d of increasing size $s_t = 2^{5.5+0.5 \cdot t}$ with $t = (1, 2, 3, \ldots, T)$, and T being a parameter set by the user such that $1 \leq T \leq 2 \cdot (\lfloor \log_2 |d_n| \rfloor - 5.5)$. This ensures that the biggest data sample never exceeds the available amount of training data. The samples follow a geometric increase, as suggested by [113]. When using a higher value for T, larger samples are calculated, presumably yielding more accurate estimates at the expense of higher run times.

Figure 6.1 shows learning curves of all model types introduced in Chapter 2.4. The x-axis is displayed on a logarithmic scale, to show the geometrical increase in sample sizes. It shows some typical learning curve behaviour. First, when presented with more data, the classifiers typically perform better. However, this is not always the case. Sometimes a classifier handles a new batch of data not so well, and accuracy decreases. Furthermore, there is also a trend of diminishing increases. At the begin-

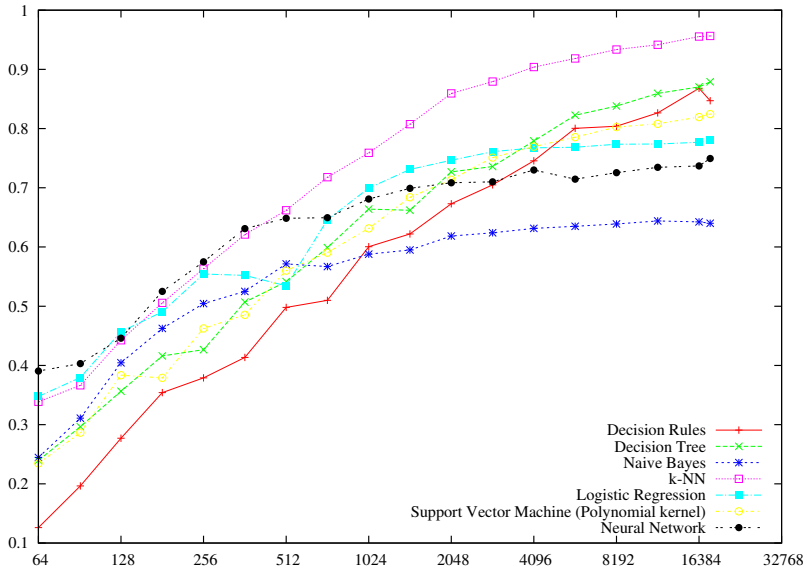


Figure 6.1: Learning curves on the 'letter' dataset.

ning the accuracy gain of adding more data is considerable, whereas at some point it flattens out. This is especially visible for models like Naive Bayes and Logistic Regression. Lastly, as already noted by [86], learning curves can cross. This also happens quite often for small samples, but less often for bigger samples.

The distance between two datasets d_i and d_j can be determined using the following function [86]:

$$dist(d_i, d_j, a_p, a_q, T) = \sum_{t=1}^{T}(P_{p,i,s_t} - P_{p,j,s_t})^2 + \sum_{t=1}^{T}(P_{q,i,s_t} - P_{q,j,s_t})^2 \tag{6.3}$$

This distance function is related to the Euclidean distance. It gives a measure of how similar two datasets are, based on the learning curves of the two classifiers. Other work proposes a distance function that measures the Manhattan distance between learning curves, but experiments show that the difference in performance between these variants is negligible [87].

Using either of these distance functions, k nearest datasets can be identified, and from the performance of both classifiers on these datasets we can predict which of the two will likely perform better on the new dataset. Controversially, it has been remarked that as the number of used samples increases, the performance of this technique decreases [86]. The authors of [86] speculate that the learning curves on the

nearest datasets are still not similar enough, and propose *Curve Adaptation*, a technique that can adapt retrieved curves to the learning curves on the new dataset. This is done because some datasets are simply harder to model, hence the whole curve will be higher or lower, and that is being corrected for. In order to adapt a learning curve of classifier a_p on dataset d_r to dataset d_i, all points of the prior learning curve are multiplied by a coefficient:

$$f(d_i, d_r, a_p, T) = \frac{\sum_{t=1}^{T}(P_{p,i,s_t} \cdot P_{p,r,s_t} \cdot s_t^2)}{\sum_{t=1}^{T}((P_{p,r,s_t})^2 \cdot s_t^2)} \tag{6.4}$$

The resulting coefficient f can be used to scale the performance of the retrieved learning curve of dataset d_r to match the partial learning curve on dataset d_i, making the final points of d_r (that are not available in d_i) more realistic.

A novel optimization that could potentially improve performance is the *Smaller Sample* technique. As not all datasets are of the same size, it is possible that a retrieved dataset has a bigger size than the new dataset, which might give an unfair advantage to particular learners. Suppose that the retrieved dataset contains a high number of observations, and from a certain sample size on the learning curve of one algorithm outperforms all other algorithms. If the new dataset is much smaller than the retrieved dataset, this information might be irrelevant and potentially obfuscates the prediction for the new dataset. In that case it might be beneficial to use the performance of the classifiers at a sample size close to the full size of the new dataset. More formally, when reasoning over the performance of a given classifier a on dataset d_{new} based on a retrieved dataset d_r, if $|d_{new}| < |d_r|$, it might be more informative to use the performance of algorithm a on a subset of d_r, rather than the full dataset d_r.

Algorithm 6.1 shows the full method in detail. It requires the new dataset as input, and values for parameters k (number of similar datasets to retrieve) and T (number of samples available to build the partial learning curve), and boolean parameters indicating whether to use the Curve Adaptation and Smaller Sample technique. The while-loop starting on line 3 identifies the most promising classifier left in A (lines 4–29), appends this classifier to the final ranking R (line 30) and removes it from the pool of remaining classifiers to rank.

To find the most promising classifier, we set a_{best} first to an arbitrary classifier left in A. We will compare it against all a_{comp} (competing) classifiers left in A (for-loop on line 5). On line 6 we retrieve a set D of datasets on which we have recorded performance results for both classifiers (recall that d_{new} is not amongst those). Line 9 uses Equation 6.3 to retrieve the nearest dataset. Lines 12–15 show how Curve Adaptation shifts the retrieved learning curve to the partial learning curve, using Equation 6.4. Lines 16–18 show how the Smaller Sample option utilizes learning curves of a size close to the size of the new dataset. The classifier that performed best on the retrieved

Algorithm 6.1 Pairwise Curve Comparison (PCC)

Require: d_{new}, $k \in \mathbb{N}^+$, $T \in \mathbb{N}^+$, $CurveAdaptation \in \{0,1\}$, $SmallerSample \in \{0,1\}$

Initialize A as a set of all classifiers
Initialize R as empty list
while $|A| > 0$ **do**
  $a_{best} \leftarrow$ Arbitrary element from A
  for all $a_{comp} \in A : a_{comp} \neq a_{best}$ **do**
    Initialize D as the set of all datasets on which a_{best} and a_{comp} were ran
    $votesBest = votesComp = 0$
    while $votesBest + votesComp < k$ **do**
      $d_{sim} \leftarrow \arg\min_{d_i \in D} dist(d_{new}, d_i, a_{best}, a_{comp}, T)$
      $coeff_{best} = coeff_{comp} = 1$
      $samp \leftarrow \Omega$
      if $CurveAdaptation = 1$ **then**
        $coeff_{best} \leftarrow f(d_{new}, d_{sim}, a_{best}, T)$
        $coeff_{comp} \leftarrow f(d_{new}, d_{sim}, a_{comp}, T)$
      end if
      if $SmallerSample = 1$ **and** $|d_{new}| < |d_{sim}|$ **then**
        $samp \leftarrow |d_{new}|$
      end if
      if $coeff_{best} \cdot P'_{best,d_{sim},samp} > coeff_{comp} \cdot P'_{comp,d_{sim},samp}$ **then**
        $votesBest \leftarrow votesBest + 1$
      else
        $votesComp \leftarrow votesComp + 1$
      end if
      $D \leftarrow D - d_{sim}$
    end while
    if $votesBest < votesComp$ **then**
      $a_{best} \leftarrow a_{comp}$
    end if
  end for
  $R \leftarrow R + a_{best}$
  $A \leftarrow A - a_{best}$
end while
return R {Ranking of classifiers in decreasing order}

dataset (line 19) gets a vote, and the dataset is removed from the pool of available datasets. This is repeated k times, for the k nearest datasets. The classifier that has most votes is marked as a_{best}, and will be compared against the next competitor a_{comp} in the following loop iteration. Note that the algorithm potentially utilizes two different evaluation scores, denoted by P and P', but these can also be the same. The scores of one evaluation measure are used for identifying similar datasets and Curve Adaptation (i.e., the one denoted by P); the scores of the other evaluation measure are used for selecting an appropriate classifier (i.e., the one denoted by P'). This is useful because not all evaluation measures are suitable for both tasks. For example, measures that trade-off accuracy and run time (e.g., ARR, $A3R$) are very suitable for selecting appropriate algorithms, however learning curves that are built upon these measures are typically neither informative nor stable.

Because we arbitrarily select the order in which classifiers are considered, the ranking will not always be the same (the meta-algorithm is unstable). However, it assumes that classifiers that perform consistently better on similar datasets will always be ranked above their inferior competitors. Furthermore, the meta-algorithm has a start up time, as it needs to build the partial learning curves. However, this is also the case for conventional meta-learning techniques (e.g., when using landmarkers). Therefore we will not consider these additional costs in the results.

6.3.2 Active Testing

Active Testing was introduced in [88]. It is an algorithm selection method that combines grid search with meta-knowledge. It recommends a ranking of classifiers (the order in which they should be cross-validated), but it also updates the ranking of the not yet cross-validated classifiers after every test.

Active Testing iteratively chooses new promising algorithms; those that have a good possibility to outperform a current best algorithm. For each candidate algorithm, it finds the historic datasets on which the candidate algorithm outperforms the current best. This is done by the notion of relative landmarkers, which are defined as:

$$RL(a_t, a_{best}, d_i, P) = b(P_{a_t,d_i,\Omega} > P_{a_{best},d_i,\Omega}) \cdot (P_{a_t,d_i,\Omega} - P_{a_{best},d_i,\Omega}) \tag{6.5}$$

where $P_{a_k,d_i,\Omega}$ is the evaluation measure (obtained by cross-validation) of algorithm a_k ($a_k \in \{best, t\}$) on dataset d_i, and b is an indicator function returning 1 if the condition is true and 0 otherwise. Active Testing operates on the full dataset rather than learning curves or sub-samples; the Ω subscript is maintained in the notation for consistency. Basically, the relative landmakers measure the accuracy difference between algorithms on datasets where the current best algorithm was outperformed,

Algorithm 6.2 Active Testing (AT)

Require: d_{new}, $a_{best} \in A$, P
1: Initialize A as a set of all classifiers except a_{best}
2: Initialize R as list containing a_{best}
3: $P'_{best,new,\Omega} = CV(a_{best}, d_{new})$
4: **while** $|A| > 0$ **do**
5: $\quad a_{comp} = \arg\max_{a_i \in A} \sum_{d_i \in D} RL(a_t, a_{best}, d_i, P) \cdot Sim(d_{new}, d_i)$
6: $\quad A \leftarrow A - a_{comp}$
7: $\quad P'_{comp,new,\Omega} = CV(a_{comp}, d_{new})$
8: $\quad$ **if** $P'_{comp,new,\Omega} > P'_{best,new,\Omega}$ **then**
9: $\quad\quad a_{best} = a_{comp}$
10: $\quad\quad P'_{best,new,\Omega} = P'_{comp,new,\Omega}$
11: $\quad$ **end if**
12: $\quad R \leftarrow R + a_{comp}$
13: **end while**
14: **return** a_{best} {Best classifier based on measure P'}

and neglect the accuracy difference on datasets where the current best algorithm was better. For each new cross-validation test, we are interested in the method that obtained the highest relative landmarker score on datasets similar to the current one. As such, we are optimizing:

$$a_{comp} = \arg\max_{a_t \in A} \sum_{d_i \in D} RL(a_t, a_{best}, d_i, P) \cdot Sim(d_{new}, d_i) \tag{6.6}$$

where $Sim(d_{new}, d_i)$ is a measure of similarity between dataset d_{new} and dataset d_i. Several similarity measures are proposed in [88]. The method *Active Testing 0* naively assumes that all datasets are equally similar, hence its corresponding similarity function always returns 1. The method *Active Testing 1* determines the similarity based on only the last cross-validation test. If the last cross-validation test shows that a_{comp} is better than the best algorithm until that moment a_{best} (i.e., $P_{a_{comp},d_{new},\Omega} > P_{a_{best},d_{new},\Omega}$), then each dataset $d_i \in D$ that satisfies $P_{a_{comp},d_i,\Omega} > P_{a_{best},d_i,\Omega}$ is considered similar to dataset d_{new}.

Algorithm 6.2 puts this all together. It requires the user to choose a measure P, the evaluation measure that should be measured by the cross-validation test. Leite et al. [88] considered only predictive accuracy, but it is clear that any measure can be used for this, e.g., $A3R$. Note that although we are using the measure determined by P to determine the order in which we search, we ultimately evaluate all algorithms based on a measure P', which is typically predictive accuracy or area under the ROC curve. The two are not necessarily identical.

Furthermore, this method requires an arbitrary classifier to be selected first, a_{best}. In the work of [88], the top ranked algorithm from the global ranking was used. This algorithm will be cross-validated first (line 3). Lines 4–13 show a while-loop, which at each iteration removes an algorithm from A (line 6), cross-validates it (line 7) and adds it to R (line 12). Which algorithm that is, is determined using Eq. 6.6 on line 5. If that algorithm performed better than the best algorithm so far, it is considered the new best algorithm (line 9–10). Finally, on line 14 the best algorithm found is returned. Note that we do not need to return a ranking, as the algorithms are already evaluated using a cross-validation test. We are guaranteed that the returned algorithm is the best on the determined criterion.

6.3.3 Combining Accuracy and Run Time

Both Active Testing and Pairwise Curve Comparison select classifiers based on their predictive accuracy on similar datasets. Both are designed such that instead of predictive accuracy any measure can be used for this selection. Because our experiments focus on both accuracy and run time, we will experiment with $A3R$, which combines predictive accuracy and run time [1]. $A3R$ compares the run times and accuracy of two classifiers on a dataset, so it could be used directly into methods that work based on pairwise comparisons. However, in order to make it useful for methods that do not compare classifiers pairwise, and allow a fair comparison in the experiments, we define a slightly adapted version of the measure:

$$A3R'^{d_i}_{a_p} = \frac{SR^{d_i}_{a_p}}{\sqrt[r]{T^{d_i}_{a_p}}} \tag{6.7}$$

where $SR^{d_i}_{a_p}$ is the predictive accuracy (success rate) of classifier a_p on dataset d_i; $T^{d_i}_{a_p}$ is the run time of classifier a_p on dataset d_i; finally, r is a parameter controlled by the user, influencing the importance of time. Indeed, a lower value results in a higher emphasize on time. The higher the $A3R'$ score, the more suitable the classifier is on the combination of accuracy and run time.

Note that both Pairwise Curve Comparison and Active Testing can natively work with $A3R$, as they work based on pairwise comparisons. However, as this does not hold for some baseline methods, it is good to have the $A3R'$ as an alternative. It is less complex and when comparing two classifiers it ranks these the same.

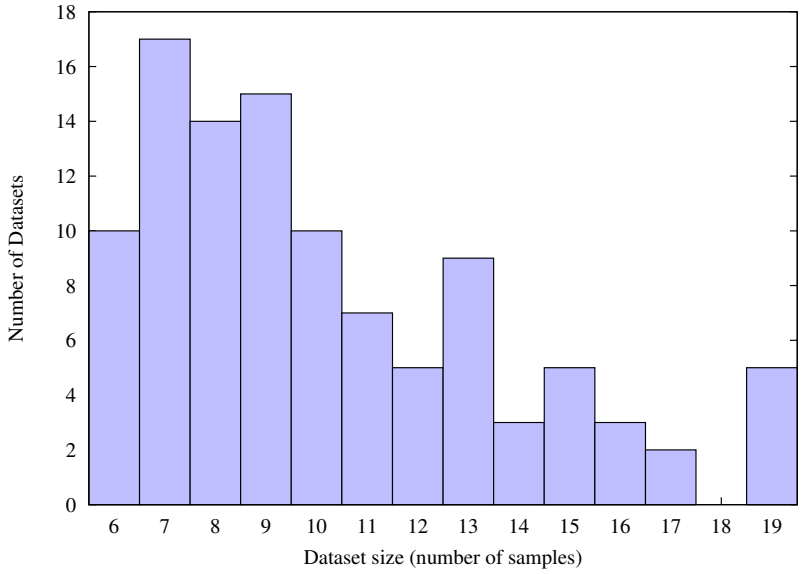


Figure 6.2: Number of datasets with maximum number of learning curve samples.

6.4 Experiments

To evaluate the proposed algorithm selection strategies, we used 30 classifiers and 105 datasets from OpenML [154]. Table 6.1 shows all datasets used in this experiment. The datasets have between 500 and 48,842 observations, and between 5 and 10,937 attributes.The size of the dataset is of importance to Pairwise Curve Comparison. In order to construct a learning curve of a given number of such samples, the dataset has to be sufficiently large (i.e., contain enough observations). In fact, to be able to construct a learning curve of 7 samples, the training set of the dataset has to contain at least 512 observations. Figure 6.2 shows the maximum number of samples that a learning curve can contain per dataset. Note that because we use 10-fold cross-validation in the experiments, a dataset actually needs 563 observations in order to guarantee a trainings set of 512 observations.

The algorithms are all from Weka 3.7.13 [61], and include all model types from Chapter 2 and all ensemble types from Chapter 3. The same algorithms are used as those used in Figure 4.10 (page 61) and Figure 4.11 (page 62). All classifiers were run on all datasets.

We will use two strong baseline methods to compare our method to. The *Best on Sample* method runs all classifiers using a given sample size, and ranks the classifiers

Table 6.1: Datasets used for the experiment.

name	obs.	atts.	cls.
irish	500	6	2
autoUniv-au7-500	500	13	5
collins	500	24	15
kc2	522	22	2
climate-model-simulation	540	21	2
cylinder-bands	540	40	2
AP_Breast_Ovary	542	10,937	2
AP_Colon_Kidney	546	10,937	2
monks-problems-3	554	7	2
monks-problems-1	556	7	2
ilpd	583	11	2
synthetic_control	600	62	6
monks-problems-2	601	7	2
AP_Breast_Kidney	604	10,937	2
balance-scale	625	5	3
AP_Breast_Colon	630	10,937	2
profb	672	10	2
soybean	683	36	19
Australian	690	15	2
credit-a	690	16	2
breast-w	699	10	2
autoUniv-au7-700	700	13	3
eucalyptus	736	20	5
blood-transfusion-serv.	748	5	2
autoUniv-au6-750	750	41	8
diabetes	768	9	2
analcatdata_dmft	797	5	6
analcatdata_authorship	841	71	4
vehicle	846	19	4
anneal	898	39	6
oh15.wc	913	3,101	10
oh5.wc	918	3,013	10
vowel	990	13	11
credit-g	1,000	21	2
autoUniv-au1-1000	1,000	21	2
autoUniv-au6-1000	1,000	41	8
oh0.wc	1,003	3,183	10
qsar-biodeg	1,055	42	2
MiceProtein	1,080	82	8
autoUniv-au7-1100	1,100	13	5
pc1	1,109	22	2
banknote-authentication	1,372	5	2
pc4	1,458	38	2
cmc	1,473	10	3
OVA_Breast	1,545	10,937	2
OVA_Colon	1,545	10,937	2
OVA_Kidney	1,545	10,937	2
OVA_Lung	1,545	10,937	2
OVA_Omentum	1,545	10,937	2
OVA_Ovary	1,545	10,937	2
OVA_Uterus	1,545	10,937	2
pc3	1,563	38	2
semeion	1,593	257	10

name	obs.	atts.	cls.
100-plants-texture	1,599	65	100
100-plants-margin	1,600	65	100
100-plants-shape	1,600	65	100
car	1,728	7	4
steel-plates-fault	1,941	34	2
mfeat-morphological	2,000	7	10
mfeat-zernike	2,000	48	10
mfeat-karhunen	2,000	65	10
mfeat-fourier	2,000	77	10
mfeat-factors	2,000	217	10
mfeat-pixel	2,000	241	10
kc1	2,109	22	2
cardiotocography	2,126	36	10
segment	2,310	20	7
scene	2,407	300	2
autoUniv-au4-2500	2,500	101	3
ozone-level-8hr	2,534	73	2
cjs	2,796	35	6
splice	3,190	62	3
kr-vs-kp	3,196	37	2
Internet-Advertisements	3,279	1,559	2
gina_agnostic	3,468	971	2
Bioresponse	3,751	1,777	2
sick	3,772	30	2
abalone	4,177	9	29
ada_agnostic	4,562	49	2
spambase	4,601	58	2
wilt	4,839	6	2
waveform-5000	5,000	41	3
phoneme	5,404	6	2
wall-robot-navigation	5,456	25	4
optdigits	5,620	65	10
first-order-theorem-proving	6,118	52	6
satimage	6,430	37	6
musk	6,598	170	2
isolet	7,797	618	26
mushroom	8,124	23	2
Gest.Ph. Segmentation Proc.	9,873	33	5
JapaneseVowels	9,961	15	9
jm1	10,885	22	2
pendigits	10,992	17	10
PhishingWebsites	11,055	31	2
sylva_agnostic	14,395	217	2
eeg-eye-state	14,980	15	2
mozilla4	15,545	6	2
MagicTelescope	19,020	12	2
letter	20,000	17	26
webdata_wXa	36,974	124	2
Click_prediction_small	39,948	12	2
electricity	45,312	9	2
tamilnadu-electricity	45,781	4	20
adult	48,842	15	2

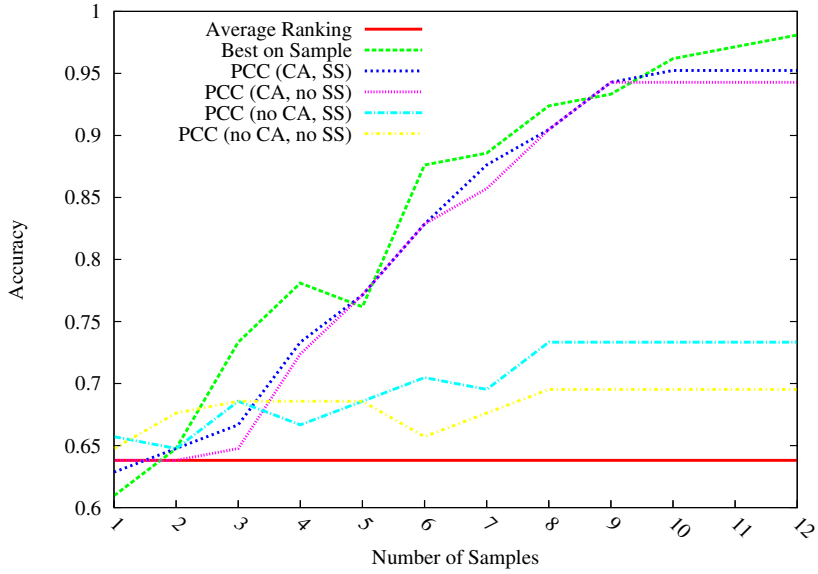
in the order of performance on that sample [107]. The *Average Ranking* method ranks the classifiers in the order of their average rank on previously seen datasets [19, 88]. The Average Ranking method represents the typical approach that human experts in machine learning wield. For each dataset, a set of favourite algorithms is selected and tried in a static order. Both baseline methods have proven to be quite accurate in previous studies.

Chapter 6.4.1 describes an experiment that focuses solely on predicting the best classifier; here we attempt to reproduce the results obtained by [87] using a larger number of datasets and classifiers. In Chapter 6.4.2 we show how the meta-algorithms perform when predicting a ranking of classifiers in Loss space. Chapter 6.4.3 shows how the meta-algorithms perform in Loss Time space. Chapter 6.4.4 describes our main contribution, empirically evaluated on both accuracy and run times. This method yields significant improvements while trading of accuracy and run time.

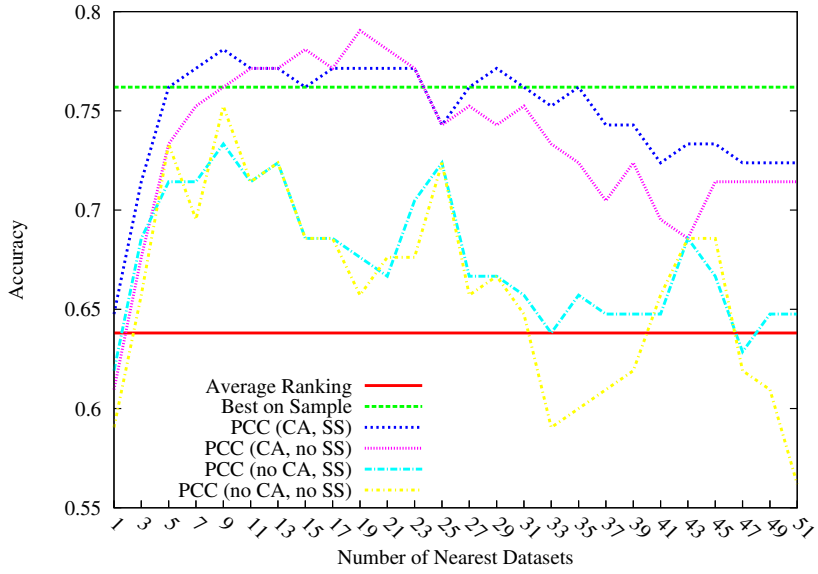
6.4.1 Predicting the Best Classifier

In the first experiment we aim to establish how well the meta-algorithm performs when the task is just to recommend the best available classifier. A recommendation is considered correct if there was no statistically significant difference between the absolute best classifier and the recommended classifier (similar to the evaluation by [87]). It uses predictive accuracy as the evaluation measure to identify similar datasets and select the best classifier (i.e., $P = P' = accuracy$). Pairwise Curve Comparison has several parameters. Most importantly, T (the number of samples used to construct the learning curves) and k (the number of nearest datasets to be identified). Furthermore, we explore the effect of Curve Adaptation (CA) and the Smaller Sample technique (SS) by comparing meta-algorithms having these options enabled against meta-algorithms having these options disabled.

Figure 6.3a shows the effect of varying the size of the learning curves. The x-axis shows the value of T (number of samples used); the y-axis shows how often a given meta-learner predicted the best algorithm or an algorithm that was statistically equivalent to the best one. Note that not all the datasets allow for the construction of learning curves containing 7–12 samples, therefore sometimes predictions are made based on a smaller learning curve than the setup actually allowed. It can be seen that for most methods, using more samples results almost consistently in better performance, as is expected. All methods outperform the Average Ranking baseline already with a small number of samples, i.e., the sample-based techniques do not need much computational effort to perform better than the Average Ranking. It is clear that the Best on Sample and Pairwise Curve Comparison with Curve Adaptation perform clearly better, even when using only a small learning curve. There are some



(a) Varied value T, fixed $k = 17$



(b) Varied value k, fixed $T = 5$

Figure 6.3: Performance of meta-algorithm on predicting the best classifier.

drops in performance, which can probably be attributed to characteristics of the specific datasets used (e.g., dimensionality). When the number of attributes of a dataset is in the same order or exceeds the number of observations, it becomes hard to learn from it and some (base-)classifiers might exhibit unstable behaviour.

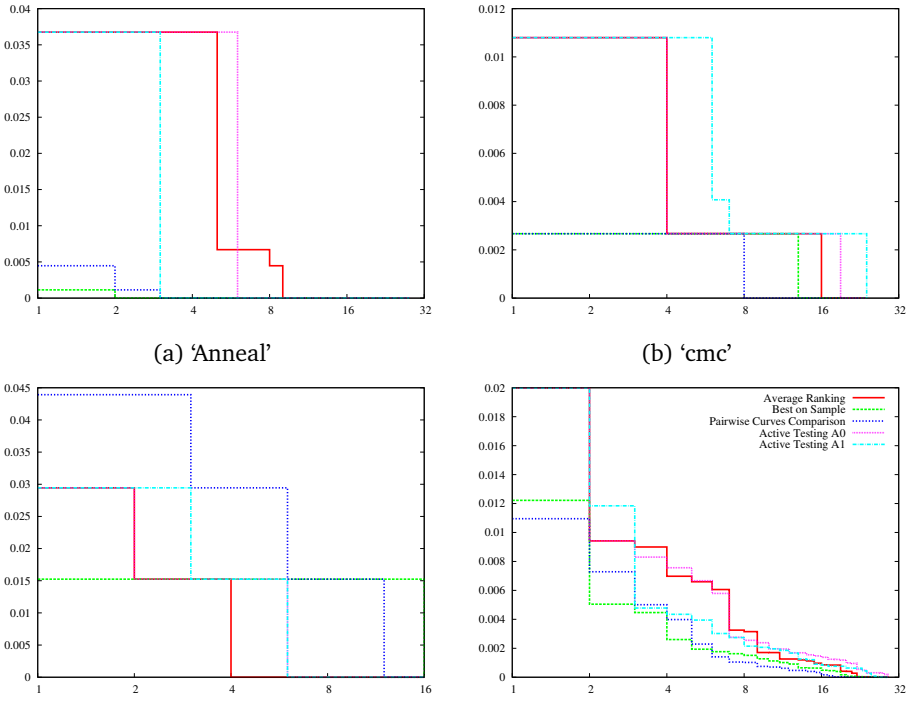
Figure 6.3b shows the effect of varying the number of nearest neighbours. In this setting, the number of samples for the learning curve T was set to 5, as this seems to be a reasonable small number. Hence, all predictions are made based on learning curves containing 5 samples, with the largest sample consisting of $2^{5.5+0.5 \cdot 5} = 256$ observations. Results obtained with higher values for T are less interesting, as the time required for running the classifier on the consecutive samples increases. Average Ranking remains constant, as it does not use samples nor identifies the nearest datasets. Setting k around 17 seem very suitable in this case, but presumably this depends on the size of the meta-dataset. Setting this value too low might lead to instable behaviour, whereas setting it too high might result in including many datasets which are not similar enough.

The Active Testing variants are not shown in both figures, as these are sequential methods that solely focus on the ranking of classifiers. It is undefined which classifier is advised first (see Algorithm 6.2). In our experiments, the classifier with the best Average Ranking is selected, so the accuracy of the first prediction of Active Testing would always be exactly the same as the Average Ranking method.

Both figures show similar trends. Best on Sample dominates the other techniques in most of the cases, even though this method is rather simple. Furthermore, both Pairwise Curve Comparison instances using Curve Adaptation (CA) outperform the instances without Curve Adaptation. Smaller Sample (SS) also seems to improve the prediction quality, although the difference is less prominent. In all, both Best on Sample and Pairwise Curve Comparison obtain very reasonable performance, advising a (statistically) best or equally good classifier in more than 75% of the cases, already when using a learning curve consisting of just 5 samples.

6.4.2 Ranking of Classifiers

Sometimes, recommencing the best base-classifier in 75 per cent of the cases is not good enough, hence we need to use a different approach. When the recommended base-classifier does not perform well enough, an alternative should be at hand. Rather than recommending a single classifier, a ranking should be created, ordering the classifiers on their likelihood of performing well on the dataset. This way, the user can make an informed decision about which models to try based on the available time and resources. The standard approach to evaluate such a ranking is to compute the Spearman Correlation Coefficient [144]. However, it has a drawback: it penalizes every



(a) 'Anneal' (b) 'cmc'

(c) 'Electricity' (d) Average over all datasets

Figure 6.4: Loss Curves. The x-axis shows the number of tests, the y-axis shows the loss.

wrongly ranked classifier equally, whereas we are mostly interested in classifiers at the top of the ranking. Furthermore, we do not care at all about incorrect ranked classifiers after the best one has been identified (although when applying the method in practise, we will not know when we have found the best method).

An alternative approach is to use Loss Curves as done in, e.g., [88]. The authors define *loss* to be the difference in accuracy between the current best classifier and the global best classifier. In order to find the global best classifier on a dataset, we evaluate all classifiers on this dataset in a certain order, for example by going down the ranking. A Loss Curve plots the obtained loss against the number of classifiers that have been tested. The goal is to find a classifier that has a low loss in relatively few tests.

In the following experiments, Pairwise Curve Comparison was run with $T = 5$ and $k = 17$. Likewise, Best on Sample was run with $T = 5$ (the sample on which base-

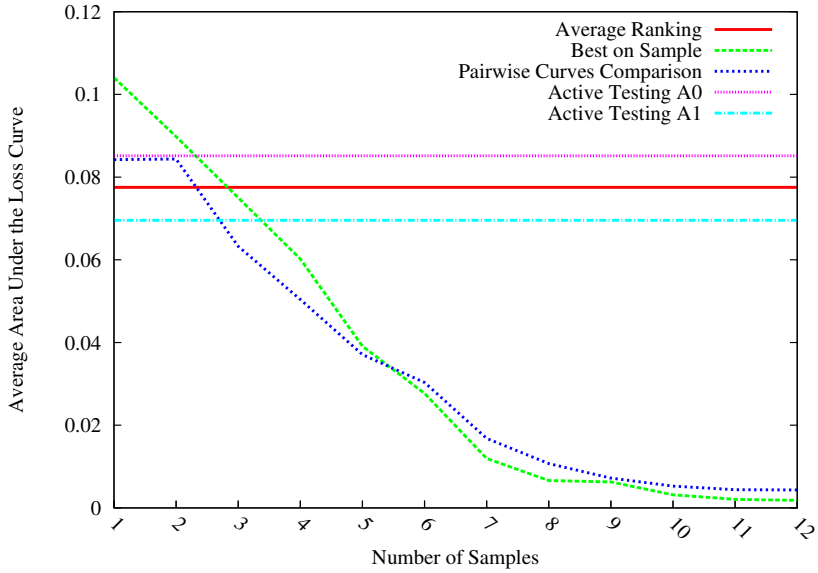


Figure 6.5: Average Area Under the Loss Curves for various meta-algorithms.

classifiers were evaluated contained 256 observations). Figure 6.4 shows loss curves of various strategies on some datasets. A loss curve is a visual evaluation measure showing how well strategies work after evaluating a given number of algorithms (tests). Similar to ROC Curves, for which commonly the Area Under the ROC Curve is calculated, we also can calculate the *Area Under the Loss Curve*, in which low values are preferred over high values. Although this measure is less informative than the Loss Curve itself, it can be used to objectively compare various meta-algorithms. Usually, this is repeated over many datasets and an average Area Under the Loss Curve is reported, as is done in [127]. Sometimes an area of interest is defined, and the Area Under the Loss Curve for that interval is calculated [3]. This is useful when there is a specific budget (expressed in number of tests) for the meta-algorithm.

Figure 6.5 plots the effect of the learning curve size on the Area Under the Loss Curve. In order to not overload the figure, we omit the Pairwise Curve Comparison instances without Curve Adaptation or the Smaller Sample option. Pairwise Curve Comparison and Best on Sample techniques dominate the other techniques. Clearly, these are the only techniques that benefit from increasing the learning curve size. The other methods are by definition not influenced by this. Active Testing A1 outperforms the Average Ranking method. It can only compete with the sample-based methods that use a very small learning curve.

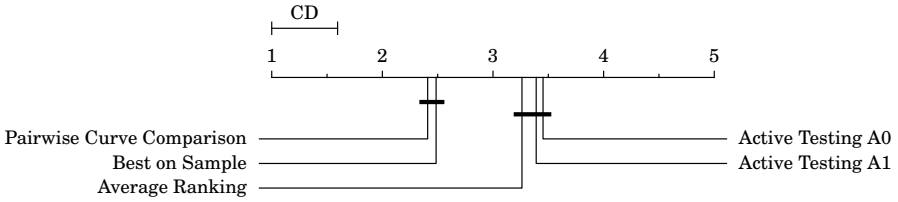


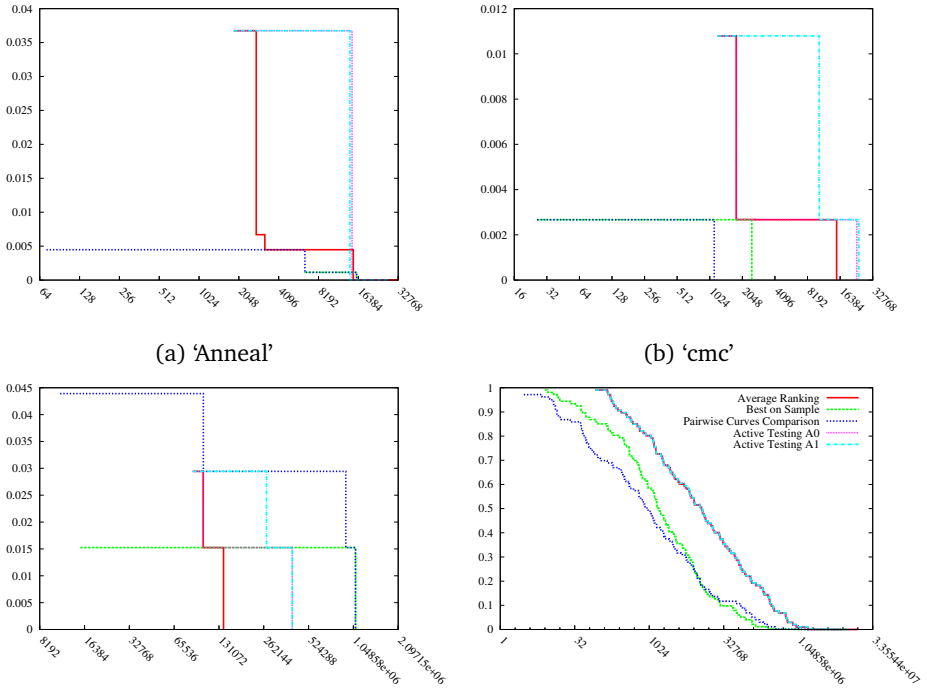
Figure 6.6: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Curve scores. Classifiers are sorted by their Average Ranking (lower is better).

Although interesting for observing general trends, we should be careful with drawing conclusions from the values from this plot: it can be dominated by outliers. Therefore, we also rank the meta-algorithms by their Area Under the Loss Curve per task, and calculate their average ranks over these. This shows which meta-algorithm performs best on most datasets, and enables us to do a statistical test over it, such as the Friedman test with post-hoc Nemenyi test. Figure 6.6 shows the result of the Nemenyi test. Pairwise Curve Comparison and Best on Sample perform statistically equivalent. The average ranks of these techniques are quite similar. Also Average Ranking, Active Testing A0 and Active Testing A1 perform statistically equivalent. Contrary to the results depicted in the Average Loss Curve (Figure 6.4d), it can be seen that the difference between Active Testing A1 and Active Testing A0 is minuscule. Because of the aforementioned reasons, conclusions based on the statistical test should have precedence over conclusions based on the average loss curve.

6.4.3 Loss Time Space

Loss Curves assume that every test will take the same amount of time, which is not realistic. For example, Multilayer Perceptrons take longer to train than Naive Bayes classifiers. Therefore, it is better to use *Loss Time Curves*, which plot the average loss against the time needed to obtain this loss. It describes how much time is needed on average to converge to a certain loss (lower is better). The faster such curve goes to a loss of zero, the better the technique is. They are commonly used in the Optimization literature (see, e.g., [74]).

Figure 6.7 shows the Loss Time Curves of various strategies on some datasets. The Loss Time Curves are drawn from the moment when the first cross-validation test has finished. As can be seen by the Loss Time Curves from Figure 6.7d, also in Loss Time space both Best on Sample and Pairwise Curve Comparison dominate the other techniques.



(a) 'Anneal' (b) 'cmc'

(c) 'Electricity' (d) Average over all datasets

Figure 6.7: Loss Time Curves. The x-axis shows the time in milliseconds, the y-axis shows the loss.

Similar to the Area Under the Loss Curve, we can calculate the Area Under the Loss Time Curve. One important aspect to consider is what loss is defined before the first cross-validation test has finished. In this work we use a loss of 1 (which is the maximum possible loss), but also other values could be chosen. For example, [3] uses *default loss*, which is defined as the difference between the default accuracy of a dataset and the accuracy of the best performing algorithm on that dataset. Figure 6.8 plots the effect of the learning curve size on the Area Under the Loss Time Curve. The legend is omitted for readability, but the meta-algorithms have the same colours as in Figure 6.7. This again confirms the dominance of Pairwise Curve Comparison and Best on Sample. However, opposed to Figure 6.5, there is no general trend that Pairwise Curve Comparison and Best on Sample benefit from using more and bigger samples. One of the explanations for this is that although we are evaluating the meta-algorithms based on accuracy and run time, internally the meta-algorithms are

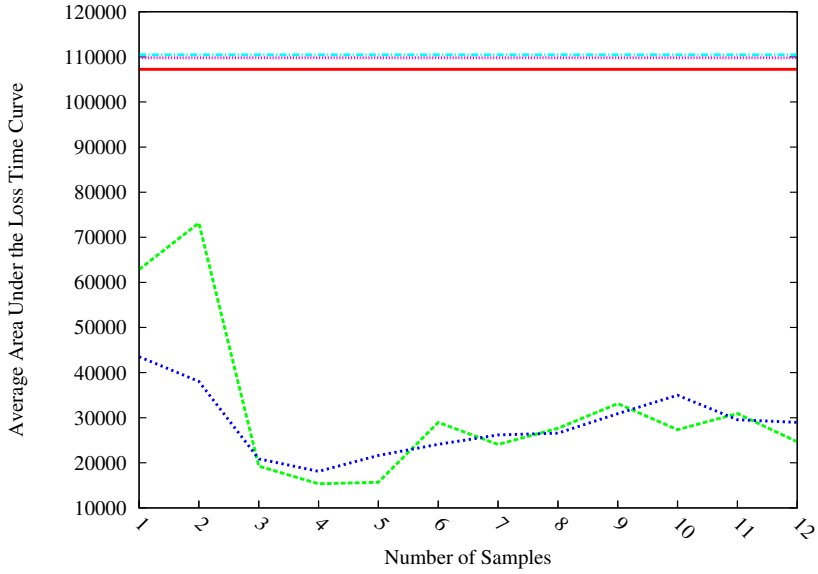


Figure 6.8: Average Area under the Loss Time Curve scores for the various meta-algorithms. The legend is omitted for readability.

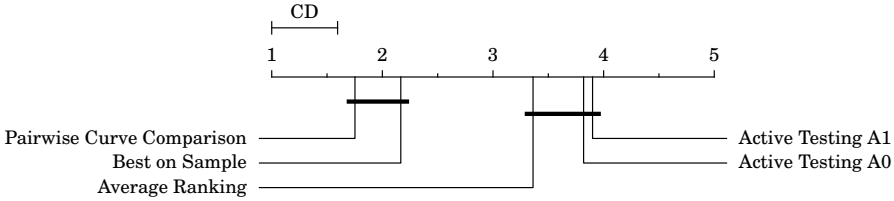


Figure 6.9: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Time Curves scores. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

not aware of this and consider only accuracy. They build a ranking solely based on accuracy, neglecting the run times, whereas high run times are punished by loss time curves.

As done before, we can rank the meta-algorithms for each dataset based on Area Under the Loss Time Curves. This enables us to do a statistical test on the performance in Loss Time space. Figure 6.9 shows the result of the corresponding Friedman with post-hoc Nemenyi test. The results seem similar to the results on the Area Under the

Loss Curves: Pairwise Curve Comparison and Best on Sample perform statistically equivalent, as well as Average Ranking, Active Testing A0 and Active Testing A1. However, much performance gain can still be obtained by making the meta-algorithms also consider run times, as we will see next.

6.4.4 Optimizing on Accuracy and Run Time

Next, our aim is to involve run times in the classifier selection process and evaluate whether this improves the performance of the meta-algorithm in Loss Time space. In this experiment, we will trade-off accuracy and run time. Recall that the meta-algorithms potentially use different evaluation measures to identify similar datasets and select classifiers. We adjust the methods to compare and select classifiers based on their $A3R'$ scores, as introduced in Chapter 6.3. Pairwise curve comparison still builds learning curves based on accuracy, but selects the most promising algorithm to test next based on a higher $A3R'$ score. Active Testing identifies similar datasets based on accuracy, but selects the most promising algorithm to test next based on a higher $A3R'$ score. Formally, evaluation measure $P = accuracy$, evaluation measure $P' = A3R'$. This way, decent classifiers that require a low run time are preferred over better classifier that require a high run time. The baseline methods can be adapted in a similar way such that they select classifiers based on $A3R'$.

Figure 6.10 compares the Average Loss Time Curves obtained using $A3R'$ with the Average Loss Time Curves based solely on accuracy. For example, in Figure 6.10a the red dashed line is exactly the same as the red line depicted in Figure 6.7d. These represent the 'vanilla' version of the Average Ranking method, that solely uses accuracy to build a ranking. In contrast, the solid red line is the version of the Average Ranking method that considers both accuracy and run time, by means of *A3R'*. The gain in performance is eminent. All methods using $A3R'$ converge to an acceptable loss orders of magnitude faster than the ones based on solely accuracy. For example, Pairwise Curve Comparison with $A3R'$ converges on average in 5 seconds to a loss of less than 0.001, whereas vanilla Pairwise Curve Comparison takes on average 454 seconds for this. This can be very useful in practise, when data needs to be processed at high speed.

Again, we do a Friedman with post-hoc Nemenyi test, based on the scores for the Area Under the Loss Time Curves per task. Figure 6.11 shows the results. All the meta-algorithms that rely on *A3R'* to construct the ranking perform statistically significant better than their vanilla counterparts. Furthermore, all meta-algorithms that construct a ranking based on *A3R'* perform better than the meta-algorithms that just use accuracy. The average ranking method scores surprisingly well in practise, which is confirmed by a similar study [3].

6.5 Conclusion

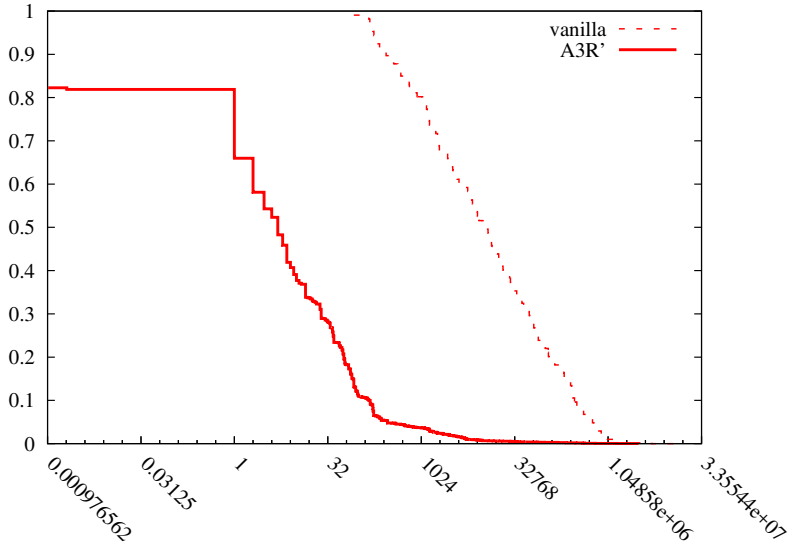
This chapter addresses the problem of algorithm selection under a budget, where multiple algorithms can be run on the full dataset until a given budget expires. This budget can be expressed as the number of cross-validation tests; in that case the user can only select a limited number of algorithms and parameter settings to evaluate. The budget can also be expressed in time, where there is a certain amount of time in which various algorithms with multiple parameter settings can be evaluated, after which the best must be selected.

We have extended the method presented in [87] such that it generates a ranking of classifiers, rather than just predicting the single best classifier. The ranking suggests in which order the classifiers should be tested. Based on such ranking a loss curve can be constructed, showing which meta-algorithms perform best after a given number of tests. In order to objectively compare various meta-learning algorithms and to enable statistical tests, the Area Under the Loss Curve can be calculated. Interestingly, a simple and elegant baseline method called Best on Sample performs equally well in our experiments, selecting a good classifier using only a few tests.

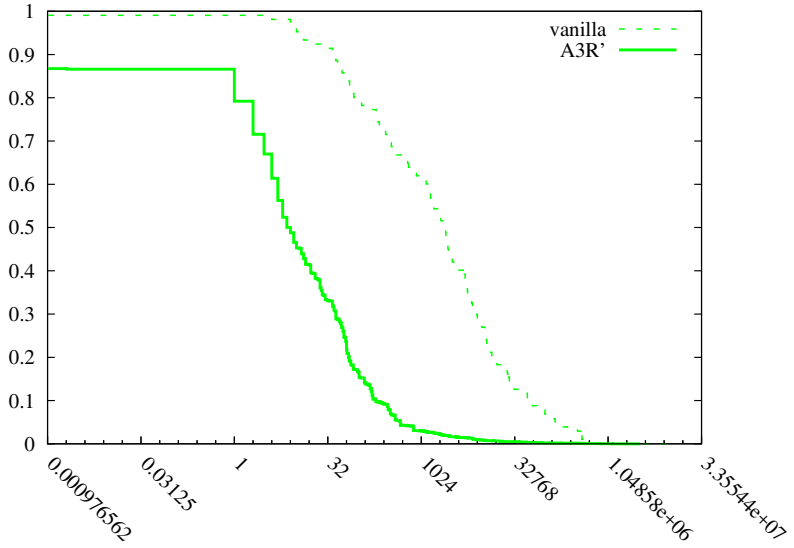
However, when tested in the more realistic setting where the budget is expressed in time, rather than a number of tests, the performance of meta-algorithms becomes unpredictable. This was reflected in Figure 6.8, where there was no general trend that Pairwise Curve Comparison and Best on Sample benefit from using more and bigger samples.

When evaluating in the time budget setting, the meta-algorithms should be aware of the run time of the classifiers. A measure such as *A3R'* (which trades off accuracy and run time) can provide this, although other measures are also capable of doing so as well (e.g., ARR). We evaluated all the meta-classifiers based on the Area Under the Loss Time Curve. The meta-algorithms using *A3R'* consistently outperformed the meta-algorithms that did not use this measure. This suggests that *A3R'* is very suitable for algorithm selection applications with a limited time budget. Apparently, it is better to try many reasonable (and fast) classifiers that a few potentially very good (but expensive) classifiers.

These results are not unexpected; when using an evaluation measure that partly considers run time (as the Area Under the Loss Time Curve does) the meta-algorithms should be aware of the run time of the algorithm configurations. Using the novel algorithm selection criterion, a reasonable performing classifier was typically selected orders of magnitude faster than otherwise. Recall that Pairwise Curve Comparison with $A3R'$ converges on average in 5 seconds to a loss of less than 0.001, whereas vanilla Pairwise Curve Comparison takes on average 454 seconds for this; a speed-up of almost factor 100.

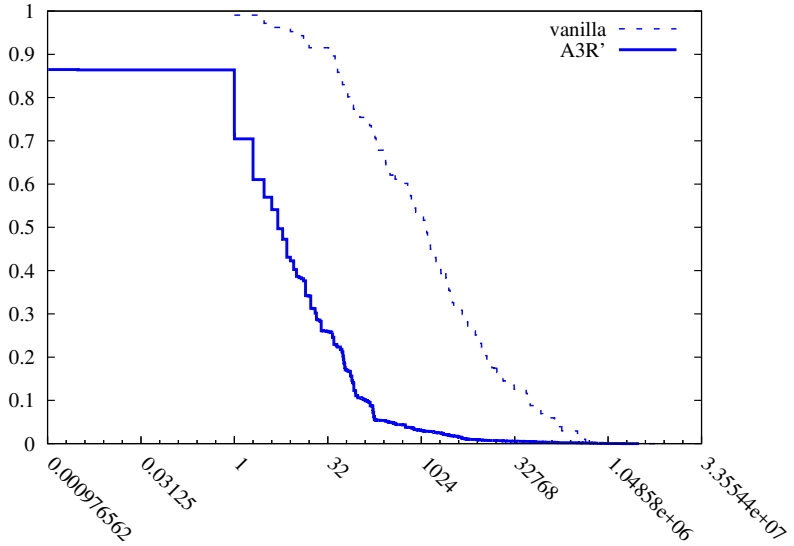


(a) Average Ranking

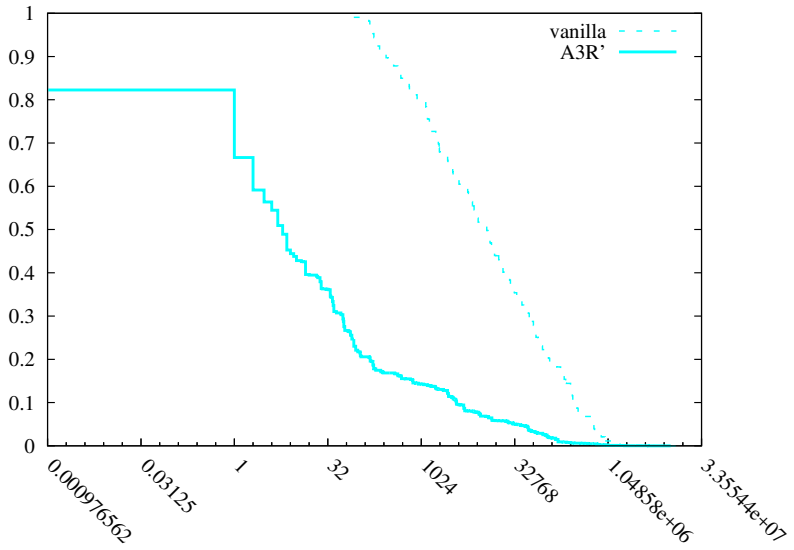


(b) Best on Sample

Figure 6.10: Loss Time Curves. The x-axis shows the time in milliseconds, the y-axis shows the loss.



(c) PCC



(d) Active Testing A1

Figure 6.10: Loss Time Curves. The x-axis shows the time in milliseconds, the y-axis shows the loss (continued).

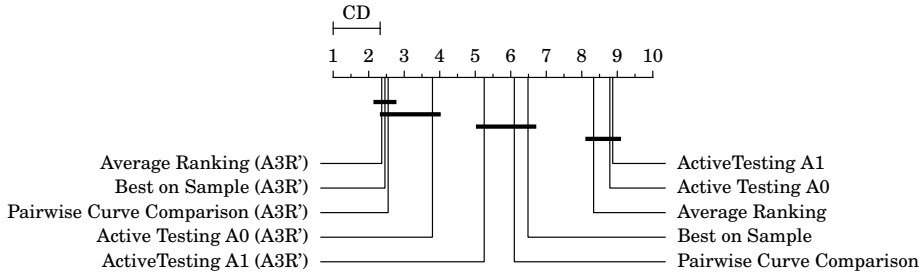


Figure 6.11: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Time Curve scores. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent. The parameters are fixed to $t = 5$, $k = 17$.

Future work should focus on applying this technique on the full meta-dataset of OpenML. Currently, OpenML already contains many datasets from various domains, e.g., data streams, text mining datasets and QSAR datasets. In this work we carefully selected a set of datasets to perform the meta-learning tasks on. However, by definition this approach embraces 'the strong assumption of Machine Learning' (as defined by Giraud-Carrier and Provost [60]). The strong assumption of Machine Learning is that the distribution of datasets that we will model is explicitly or implicitly known, at least to a useful approximation (see also Chapter 3). To the best of our knowledge, there is currently no work in Machine Learning that does not make this assumption. However, Machine Learning and meta-learning do not require such strong assumptions. In other words, one very promising area of future work would be setting up a meta-learning experiment that involves all datasets from OpenML. Such experiment would face many challenges, as one would be dealing with data and classifiers from various domains, but the rewards could be high. Trading off accuracy and run time using $A3R'$ might be a key aspect to this. This would be the first research that convincingly demonstrates the true power of meta-learning, without making any strong assumptions.

7 Conclusions

7.1 Open Machine Learning

We have introduced OpenML, an online platform on which researchers can share and reuse large amounts of collaboratively generated experimental data. OpenML automatically stores and indexes important meta-data about the experiments for reproducibility and further study. For datasets, data about the attributes and standard meta-features are stored. Upon these datasets, well-defined scientific tasks can be created. These provide a formal description about the given inputs and required outputs for solving it. This makes the uploaded results objectively comparable.

For the uploaded algorithms, all parameters, their data types and default values are registered. This way, it is possible to compare the performance of various algorithms, but also various parameter settings of the same algorithm. Furthermore, OpenML is integrated in popular Machine Learning workbenches and programming languages, making it possible to share algorithms and experiments with a few lines of code, or a single click of a button.

For all experiments, the exact algorithms, the inputs (such as parameter settings) and the outputs (models, predictions on test set) are stored. This makes it possible to reuse this information and learn from the past. The experimental data answers many questions about the interplay between data and algorithms, for example

- what is the best algorithm for a certain data set?
- how does a given data property influence the performance of an algorithm?
- what is the effect of a given parameter on the performance of an algorithm?
- which parameters are influencing predictive performance the most?

- which pairs of algorithms have a similar (or different) prediction behaviour?

Many of these research questions require the setup of time and computation-intensive experiments, while these can be answered on the fly when adopting this collaborative approach.

7.2 Massively Collaborative Machine Learning

We demonstrated the power of this collaborative approach by means of two large scale studies. The first study covered the data stream setting, where classifiers are continuously trained and evaluated on a stream of new observations. We ran a wide range of classifiers over all data streams in OpenML, and built meta-models to predict for each (window of) instances which classifier would work best on it. Indeed, dynamically switching at various points in the stream between various heterogeneous classifiers potentially results in a better accuracy than individual classifiers could achieve. This technique (the Meta-Learning Ensemble) indeed outperformed all individual classifiers and is competitive with state-of-the-art ensembles using many more models. Quite surprisingly, an even simpler technique that measured which of the classifiers performed best on a previous window (BLAST) outperformed all other approaches based on this idea. We introduced two variants of measuring the performance of classifiers on previous data, one approach using a fixed window and one approach based on fading factors. Furthermore, we built a clustering upon the instance-based predictions of all data stream classifiers, gaining insight in which classifiers make similar predictions (Figure 4.14 on page 69). We used OpenML to scale up data stream studies to cover 60 data streams, which is to the best of our knowledge the largest data stream study so far.

The second study covered conventional batch data, and leveraged learning curve information about algorithms. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size. OpenML contains many of these. The meta-algorithms leveraged learning curves up till a certain size, which is usually much faster than running a set of algorithms on the full dataset. Based on the performance of an algorithm on the first samples of a learning curve, assumptions can be made about the performance on the whole dataset. Within a certain budget, the most promising classifiers can be tested using a cross-validation procedure. The budget can be expressed in either an amount of cross-validation tests, or run time. In the latter case, it proved very fruitful to select classifiers based on a trade-off between accuracy and run time. If one is willing to settle for an algorithm that is almost as good as the absolute best algorithm for that dataset, the costs of finding an appropriate algorithm can be decreased by orders of magnitude. In this study, OpenML was

used as an experiment repository: the datasets and tasks that were used take many resources (time and memory) to model, so the only way to comprehensively research the proposed techniques is by reusing results that are collaboratively generated.

7.3 Community Adoption

More researchers have already adopted OpenML, as can be seen by the increasing amount of uploaded experiments. We mention some noteworthy studies, which is just a small selection of successful examples. The work of Feurer et al. [43] specializes in algorithm selection for Machine Learning by means of Sequential Model-based Bayesian Optimization. As mentioned in Chapter 3, the performance of Sequential Model-based Bayesian Optimization depends on the quality of the initial evaluation points. In order to find good initial evaluation points, the meta-knowledge in OpenML is used.

The work of Olier et al. [99] focuses on automated drug discovery. Given a protein that is critical to a pathogen (e.g. a virus or parasite), chemists are interested to know which molecules (drugs) can successfully inhibit that pathogen. This information is stored in QSAR datasets, which link the structural properties of the drugs to their activity against the protein. Machine Learning techniques can learn this relationship, but it is not clear which techniques will work best on a given QSAR dataset. The authors investigated whether a meta-algorithm can learn which techniques work well on any given QSAR dataset. All results (datasets and algorithm evaluations) are available on OpenML.

The work of Post et al. [111] uses OpenML to make general claims about common Machine Learning assumptions. In this work, the authors investigated which classifiers benefit from feature selection. Quite surprisingly, feature selection seldom led to a statistical significant improvement in performance. The authors speculate that while this might be the case, the set of datasets might be biased. As all the datasets in OpenML come from Machine Learning problems, chances are high that these already experienced some form of pre-processing. Applying these techniques on more raw data might show different results.

7.4 Future Work

With these and many other ongoing projects, there is much room for future work. Machine Learning learning literature contains a lot of 'folk knowledge' and 'folk wisdom' [38], but in order to be scientifically correct we need proper theoretical or experimental results to back these up. Much of this folklore can be confirmed or rejected

by means of large scale experimentation. Investigating questions like “is data preprocessing more important than proper algorithm selection?”, “are non-linear models really better than linear models?” and “how much additional training effort do non-linear models need to outperform their linear counterparts?” would spark interesting discussions within the community.

Another obvious possibility is to do a large scale benchmark of Machine Learning algorithms. Recently, a particular benchmark study attracted lots of attention [42], but was also criticized for various reasons [160]. In order to do proper benchmarking, the datasets, algorithms and performance space need to be properly defined. Furthermore, the relevant algorithm parameters need to be properly tuned, using for example Bayesian Optimization or Random Search. We believe that with OpenML, the infrastructure for a proper benchmark study is available, making it possible to compare algorithms across various Machine Learning toolboxes and making the results interactively available.

To conclude, there is the issue of meta-learning and optimization. As was argued in Chapter 3, there are basically two approaches to algorithm selection. The meta-learning approach learns from prior experiments, and recommends a set of classifiers based on these. The search approach experiments with intelligently trying out various classifiers (and parameter settings), but neglects the vast amount of experimental results already available. Earlier attempts to combine the two resulted in interesting ideas and solid results [43, 88], however there has been little follow-up. One possible explanation for this might be the fact that it combines two sources of knowledge that are computationally expensive to acquire, and complex to understand. OpenML partly abates both difficulties: knowledge of past experiments is available by querying the database and will lead to more understanding of both techniques. Successfully combining these two paradigms has the potential to convincingly push the state of the art of both fields of research.

Bibliography

[1] S. M. Abdulrahman and P. Brazdil. Measures for Combining Accuracy and Time for Meta-learning. In *Meta-Learning and Algorithm Selection Workshop at ECAI 2014*, pages 49–50, 2014.

[2] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren. Algorithm selection via meta-learning and sample-based active testing. In J. Vanschoren, P. Brazdil, C. Giraud-Carrier, and L. Kotthoff, editors, *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection (MetaSel)*, number 1455 in CEUR Workshop Proceedings, pages 55–66, Aachen, 2015.

[3] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning, Special Issue on Metalearning and Algorithm Selection*, forthcoming, 2016.

[4] D. W. Aha. Generalizing from case studies: a case study. In *Proceedings of the ninth international workshop on Machine learning*, pages 1–10, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[5] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000.

[6] C. Apté and S. Weiss. Data mining with decision trees and decision rules. *Future generation computer systems*, 13(2):197–210, 1997.

[7] M. Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

[8] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[10] J. Beringer and E. Hüllermeier. Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6):627–650, 2007.

[11] A. Bifet and R. Gavalda. Learning from Time-Changing Data with Adaptive Windowing. In *SDM*, volume 7, pages 139–148. SIAM, 2007.

[12] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII*, pages 249–260. Springer, 2009.

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

[14] A. Bifet, G. Holmes, and B. Pfahringer. 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, 2010.

[15] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer. Ensembles of restricted hoeffding trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3 (2):30, 2012.

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

[17] A. L. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992.

[18] L. Bottou. Stochastic Learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

[19] P. Brazdil and C. Soares. A Comparison of Ranking Methods for Classification Algorithm Selection. In *Machine Learning: ECML 2000*, pages 63–75. Springer, 2000.

[20] P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Machine Learning: ECML-94*, pages 83–102. Springer, 1994.

[21] P. Brazdil, C. Soares, and J. P. Da Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3): 251–277, 2003.

[22] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2009.

[23] L. Breiman. Bagging Predictors. *Machine learning*, 24(2):123–140, 1996.

[24] L. Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001.

[25] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[26] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[27] C. E. Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 17–24, 1993.

[28] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. 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, 2013.

[29] J. Carpenter. May the best analyst win. *Science*, 331(6018):698–699, 2011.

[30] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

[31] D. M. Chickering. Learning bayesian networks is np-complete. In *Learning from data*, pages 121–130. Springer, 1996.

[32] W. W. Cohen. Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, pages 115–123, 1995.

[33] P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, P. Preston, and A. Srinivasan. Ripple down rules: Turning knowledge acquisition into knowledge maintenance. *Artif. Intell. Med.*, 4(6):463–475, Dec. 1992.

[34] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

[35] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[36] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[37] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[38] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[39] P. Domingos and G. Hulten. Mining High-Speed Data Streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[40] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

[41] H. J. Escalante, M. Montes, and L. E. Sucar. Particle Swarm Model Selection. *The Journal of Machine Learning Research*, 10:405–440, 2009.

[42] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[43] M. Feurer, J. T. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, 2015.

[44] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[45] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[46] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, pages 148–156. Morgan Kaufmann, 1996.

[47] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6:161–182, 1991.

[48] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

[49] J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[50] J. H. Friedman. On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[51] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[52] J. Fürnkranz and J. Petrak. An Evaluation of Landmarking Variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68, 2001.

[53] J. Gama and P. Brazdil. Cascade Generalization. *Machine Learning*, 41(3): 315–343, 2000.

[54] J. Gama and P. Kosina. Recurrent concepts in data streams classification. *Knowledge and Information Systems*, 40(3):489–507, 2014.

[55] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2004.

[56] J. Gama, P. Medas, and R. Rocha. Forest Trees for On-line Data. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 632–636. ACM, 2004.

[57] J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in Evaluation of Stream Learning Algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM, 2009.

[58] J. Gama, R. Sebastião, and P. P. Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[59] C. Giraud-Carrier. Beyond predictive accuracy: What? Technical report, University of Bristol, Bristol, UK, UK, 1998.

[60] C. Giraud-Carrier and F. Provost. Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. In *Proceedings of the ICML-2005 Workshop on Meta-learning*, pages 12–19, 2005.

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

[62] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

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

[64] L. Hansen and P. Salamon. Neural Network Ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993–1001, 1990.

[65] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning, 2nd edition*. Springer New York, 2009.

[66] J. L. Hintze and R. D. Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

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

[68] M. W. Hoffman, B. Shahriari, and N. de Freitas. Exploiting correlation and budget constraints in bayesian multi-armed bandit optimization. *arXiv preprint arXiv:1303.6746*, 2013.

[69] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[70] H. J. Hoogeboom, W. A. Kosters, J. N. van Rijn, and J. K. Vis. Acyclic Constraint Logic and Games. *ICGA Journal*, 37(1):3–16, 2014.

[71] V. Hoste and W. Daelemans. Comparing learning approaches to coreference resolution. There is more to it than bias. In *Proceedings of the ICML'05 Workshop on Meta-learning*, pages 20–27, 2005.

[72] J. Hühn and E. Hüllermeier. Furia: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319, 2009.

[73] G. Hulten, L. Spencer, and P. Domingos. Mining Time-changing Data Streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[74] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. Murphy. Time-Bounded Sequential Parameter Optimization. In *Learning and intelligent optimization*, pages 281–298. Springer, 2010.

[75] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[76] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[77] A. Jain and D. Zongker. Feature Selection: Evaluation, Application, and Small Sample Performance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(2):153–158, 1997.

[78] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[79] T. Kealey. *Sex, science and profits*. Random House, 2010.

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

[81] R. D. King, S. H. Muggleton, A. Srinivasan, and M. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996.

[82] R. Kohavi, D. H. Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.

[83] K. K. Ladha. Condorcet's jury theorem in light of de finetti's theorem. *Social Choice and Welfare*, 10(1):69–85, 1993.

[84] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[85] J. W. Lee and C. Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

[86] R. Leite and P. Brazdil. Predicting Relative Performance of Classifiers from Samples. In *Proceedings of the 22nd international conference on Machine learning*, pages 497–503. ACM, 2005.

[87] R. Leite and P. Brazdil. Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning. In *ECAI*, pages 309–314, 2010.

[88] R. Leite, P. Brazdil, and J. Vanschoren. Selecting Classification Algorithms with Active Testing. In *Machine Learning and Data Mining in Pattern Recognition*, pages 117–131. Springer, 2012.

[89] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. *arXiv preprint arXiv:1603.06560*, 2016.

[90] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[91] I. Manolescu and S. Manegold. Performance evaluation in database research: principles and experience. *Proceedings of the International Conference on Extending Database Technology (EDBT)*, page 1156, 2008.

[92] M. Meeng and A. Knobbe. Flexible enrichment with cortana–software demo. In *Proceedings of BeneLearn*, pages 117–119, 2011.

[93] D. Michie, D. Spiegelhalter, and C. Taylor. *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X.

[94] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, and L. Wan. Heterogeneous Ensemble for Feature Drifts in Data Streams. In *Advances in Knowledge Discovery and Data Mining*, pages 1–12. Springer, 2012.

[95] P. Nguyen, M. Hilario, and A. Kalousis. Using Meta-mining to Support Data Mining Workflow Planning and Optimization. *Journal of Artificial Intelligence Research*, 51:605–644, 2014.

[96] M. Nielsen. The future of science: Building a better collective memory. *APS Physics*, 17(10), 2008.

[97] M. Nielsen. *Reinventing discovery: the new era of networked science*. Princeton University Press, 2012.

[98] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12): 1565–1567, 2006.

[99] I. Olier, C. Grosan, N. Sadawi, L. Soldatova, and R. D. King. Meta-qsar: Learning how to learn qsars. In J. Vanschoren, P. Brazdil, C. Giraud-Carrier, and

L. Kotthoff, editors, *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection (MetaSel)*, number 1455 in CEUR Workshop Proceedings, pages 104–105, Aachen, 2015.

[100] E. Ostrom. Collective action and the evolution of social norms. *The Journal of Economic Perspectives*, pages 137–158, 2000.

[101] N. C. Oza. Online Bagging and Boosting. In *Systems, man and cybernetics, 2005 IEEE international conference on*, volume 3, pages 2340–2345. IEEE, 2005.

[102] T. Pedersen. Empiricism is not a matter of faith. *Computational Linguistics*, 34: 465–470, 2008.

[103] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[104] Y. Peng, P. Flach, C. Soares, and P. Brazdil. Improved dataset characterisation for meta-learning. *Lecture Notes in Computer Science*, 2534:141–152, Jan 2002.

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

[106] A. H. Peterson and T. Martinez. Estimating The Potential for Combining Learning Models. In *In Proc. of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.

[107] J. Petrak. Fast Subsampling Performance Estimates for Classification Algorithm Selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 3–14, 2000.

[108] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Tell me who can learn you and I can tell you who you are: Landmarking Various Learning Algorithms. In *Proceedings of the 17th international conference on machine learning*, pages 743–750, 2000.

[109] B. Pfahringer, G. Holmes, and R. Kirkby. New Options for Hoeffding Trees. In *AI 2007: Advances in Artificial Intelligence*, pages 90–99. Springer, 2007.

[110] F. Pinto, C. Soares, and J. Mendes-Moreira. Towards automatic generation of metafeatures. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 215–226. Springer, 2016.

[111] M. J. Post, P. van der Putten, and J. N. van Rijn. Does Feature Selection Improve Classification? A Large Scale Experiment in OpenML. In *Advances in Intelligent Data Analysis XV*, pages 158–170. Springer, 2016.

[112] J. Priem, P. Groth, and D. Taraborelli. The Altmetrics Collection. *PLoS ONE*, 11(7):e48753, 2012.

[113] F. Provost, D. Jensen, and T. Oates. Efficient Progressive Sampling. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 23–32. ACM, 1999.

[114] P. van der Putten and M. van Someren. A bias-variance analysis of a real world learning problem: The coil challenge 2000. *Machine Learning*, 57(1):177–195, 2004.

[115] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[116] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[117] M. Radovanović, A. Nanopoulos, and M. Ivanović. Hubs in space: Popular nearest neighbors in high-dimensional data. *JMLR*, 11:2487–2531, 2010.

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

[119] J. R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65118, 1976.

[120] J. N. van Rijn. Playing Games: The complexity of Klondike, Mahjong, Nonograms and Animal Chess. Master's thesis, Leiden University, 2012.

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

[122] J. N. van Rijn and J. K. Vis. Complexity and retrograde analysis of the game dou shou qi. In *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence*, 8 pages, 2013.

[123] J. N. van Rijn and J. K. Vis. Endgame Analysis of Dou Shou Qi. *ICGA Journal*, 37(2):120–124, 2014.

[124] J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. OpenML: A Collaborative Science Platform. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–649. Springer, 2013.

[125] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Algorithm Selection on Data Streams. In *Discovery Science*, volume 8777 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2014.

[126] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Towards Meta-learning over Data Streams. In J. Vanschoren, P. Brazdil, C. Soares, and L. Kotthoff, editors, *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, number 1201 in CEUR Workshop Proceedings, pages 37–38, Aachen, 2014.

[127] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren. Fast Algorithm Selection using Learning Curves. In *Advances in Intelligent Data Analysis XIV*, pages 298–309. Springer, 2015.

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

[129] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Case Study on Bagging Stable Classifiers for Data Streams. In *Proceedings of the 24th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2015)*, 6 pages, 2015.

[130] J. N. van Rijn, F. W. Takes, and J. K. Vis. The complexity of rummikub problems. In *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC 2015)*, 8 pages, 2015.

[131] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren. On the Evaluation of Algorithm Selection Problems. In *Proceedings of the 25th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2016)*, 2 pages, 2016.

[132] L. Rokach and O. Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[133] A. L. D. Rossi, A. C. P. de Leon Ferreira, C. Soares, and B. F. De Souza. MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, 127:52–64, 2014.

[134] C. Schaffer. A conservation law for generalization performance. In *Proceedings of the 11th international conference on machine learning*, pages 259–265, 1994.

[135] R. E. Schapire. The Strength of Weak Learnability. *Machine learning*, 5(2): 197–227, 1990.

[136] J. C. Schlimmer. *Concept Acquisition Through Representational Adjustment*. PhD thesis, University of California, Irvine, 1987.

[137] J. C. Schlimmer and R. H. Granger Jr. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[138] A. Shaker and E. Hüllermeier. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing*, 150:250–264, 2015.

[139] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[140] M. R. Smith, T. Martinez, and C. Giraud-Carrier. An instance level analysis of data complexity. *Machine learning*, 95(2):225–256, 2014.

[141] K. A. Smith-Miles. Cross-disciplinary Perspectives on Meta-Learning for Algorithm Selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2008.

[142] S. Sonnenburg, M. Braun, C. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K. Muller, F. Pereira, C. Rasmussen, G. Ratsch, B. Scholkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

[143] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.

[144] Q. Sun and B. Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine learning*, 93(1):141–161, 2013.

[145] Q. Sun, B. Pfahringer, and M. Mayo. Towards a Framework for Designing Full Model Selection and Optimization Systems. In *Multiple Classifier Systems*, pages 259–270. Springer, 2013.

[146] A. S. Szalay, J. Gray, A. R. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. 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, 2002.

[147] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[148] L. Torgo. *Data Mining with R: Learning with Case Studies*. Chapman & Hall/CRC, 1st edition, 2010. ISBN 1439810184, 9781439810187.

[149] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[150] J. N. Van Rijn and J. Vanschoren. OpenML: An Open Science Platform for Machine Learning. In *Proceedings of the 22th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2013)*, 1 page, 2013.

[151] J. N. Van Rijn, V. Umaashankar, S. Fischer, B. Bischl, L. Torgo, B. Gao, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. A RapidMiner extension for Open Machine Learning. In *RapidMiner Community Meeting and Conference*, pages 59–70, 2013.

[152] J. Vanschoren and H. Blockeel. Towards understanding learning behavior. In *Proceedings of the annual machine learning conference of Belgium and the Netherlands*, pages 89–96, 2006.

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

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

[155] J. Vanschoren, J. N. van Rijn, and B. Bischl. Taking machine learning research online with OpenML. In *Proceedings of the 4th International Workshop on Big*

Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, pages 1–4. JLMR.org, 2015.

[156] J. Vanschoren, J. N. van Rijn, B. Bischl, G. Casalicchio, M. Lang, and M. Feurer. OpenML: a Networked Science Platform for Machine Learning. In *ICML 2015 MLOSS Workshop*, 3 pages, 2015.

[157] V. Vidulin, M. Bohanec, and M. Gams. Combining human analysis and machine data mining to obtain credible data relations. *Information Sciences*, 288:254–278, 2014.

[158] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[159] R. Vilalta, C. Giraud-Carrier, and P. Brazdil. *Meta-Learning*, pages 731–748. Springer US, Boston, MA, 2005.

[160] M. Wainberg, B. Alipanahi, and B. J. Frey. Are random forests truly the best classifiers? *Journal of Machine Learning Research*, 17(110):1–5, 2016.

[161] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining Concept-Drifting Data Streams using Ensemble Classifiers. In *KDD*, pages 226–235, 2003.

[162] M. Wojnarski, S. Stawicki, and P. Wojnarowski. Tunedit.org: System for automated evaluation of algorithms in repeatable experiments. *Lecture Notes in Computer Science*, 6086:20–29, 2010.

[163] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[164] M. N. Zarmehri and C. Soares. Using Metalearning for Prediction of Taxi Trip Duration Using Different Granularity Levels. In *Advances in Intelligent Data Analysis XIV*, pages 205–216. Springer, 2015.

[165] P. Zhang, B. J. Gao, X. Zhu, and L. Guo. Enabling Fast Lazy Learning for Data Streams. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 932–941. IEEE, 2011.

Dutch Summary

Veel wetenschap is gericht op het bouwen van modellen. Modellen zijn schematische (vaak versimpelde) weergaven van de werkelijkheid. Deze modellen zijn gebaseerd op waargenomen data uit het verleden, en maken vaak voorspellingen voor de toekomst (nog niet waargenomen data). De mens werkt onbewust met modellen; vrijwel alle informatie die wij tot ons nemen verwerken wij tot een model. Een klassiek Nederlands voorbeeld hiervan is het weer. Wanneer we 's ochtends uit het raam kijken en er hangen donkere wolken, zouden we geneigd zijn te denken dat het een regenachtige dag wordt. Andersom, wanneer de lucht helder blauw is, zouden we wellicht denken dat het een mooie dag gaat worden. Het model dat wij gebruiken stelt dus: "Als het bewolkt is, is er een grote kans op regen; en als de lucht helder is, is er een grote kans op een mooie dag." Op een of andere manier hebben mensen de capaciteit om ervaringen uit het verleden om te zetten in zulke modellen, en daarmee voorspellingen te maken voor de toekomst. Dit noemen we ook wel 'leren'.

Tegenwoordig zijn er veel technieken waardoor computers ook kunnen leren. Dit vakgebied heet Machine Learning, en wordt in het dagelijks leven op veel plekken toegepast. Voorbeelden hiervan zijn:

- sociale media: de computer bepaalt welke advertenties er aan een specifieke gebruiker getoond moeten worden
- aandelenmarkt: de computer bepaalt welke aandelen gekocht of verkocht moeten worden
- spamfilters: de computer bepaalt voor inkomende berichten of het spam of geen spam is

Aangezien veel van deze onderwerpen 'hot topics' zijn, is er veel onderzoek gestoken in het ontwikkelen van computer programma's die zulke modellen kunnen bouwen

(algoritmes). Er bestaan bijzonder veel van zulke algoritmes, en vaak hebben deze algoritmes ook nog verschillende opties die de werking subtiel beïnvloeden (parameters). Daarnaast is er sluitend wiskundig bewijs dat er niet één algoritme bestaat dat goed werkt op alle datasets. Dat maakt het voor eindgebruikers die deze algoritmes op hun data willen toepassen niet makkelijk om het juiste algoritme te kiezen.

Het vakgebied van 'meta-learning' heeft als doel orde in deze chaos te scheppen. Het houdt zich bezig met het in kaart brengen van welke algoritmes het goed doen op wat voor soort data. Dit wordt veelal op experimentele basis gedaan. Een veel gebruikte methodologie is om zo veel mogelijk algoritmes op zo veel mogelijk datasets uit te proberen, en op basis van de resultaten een model te leren welke op welke datasets bepaalde algoritmes goed werken (vandaar de naam: meta-learning). Hoewel dit in de praktijk goed werkt is de tijd een limiterende factor. Vaak nemen deze experimenten veel tijd in beslag, waardoor het onderzoek noodgedwongen kleinschaliger is dan wenselijk.

Om dit probleem op te lossen hebben we OpenML ontwikkeld. Dit is een online database waarop onderzoekers hun experimentele data met elkaar kunnen delen, om op die manier grootschaliger meta-learning onderzoek te kunnen doen. Wanneer eerdere experimentele resultaten opgeslagen en vrij toegankelijk beschikbaar zijn, is het niet langer nodig om deze tijdrovende experimenten op te zetten, maar kunnen vragen over de werking van algoritmes direct grondig worden beantwoord.

In dit proefschrift wordt beschreven hoe fundamentele meta-learning problemen als 'welk algoritme heeft de voorkeur op een bepaald soort data?', 'wat is het effect van een bepaalde parameter op de prestaties van een algoritme?' en 'hoe werkt een zojuist nieuw ontwikkeld algoritme ten opzichte van alle bestaande algoritmes?' met behulp van OpenML met relatief weinig moeite kunnen worden beantwoord.

English Summary

Many scientists are focussed on building models. Models are a schematical representation of a concept (often simplified). These models are based on observed data from the past, and make predictions about the future (yet unseen data). Subconsciously, we work with many models. We nearly process all information to a model. The weather is a classic Dutch example. When we observe dark clouds in the morning, we are inclined to think that it will be a rainy day. Conversely, when there is a clear blue sky, we will expect a sunny day. In this case, the model that we are using is: "If there are clouds, chances are that it is going to rain; if there is a clear sky, we might have a nice day." Somehow, we are capable of turning our experiences from the past into such models, and use these to make predictions about the future. This is called 'learning'.

There are many techniques that enable computers to learn as well. The field of research that develops such techniques is called Machine Learning. We encounter Machine Learning on a daily basis; some examples are:

- social media: the computer decides which advertisements should be shown to a specific user
- the stock market: the computer decides which stocks should be bought or sold
- spam filters: the computer determines whether an incoming message is spam

Many of these subjects are considered hot topics, hence many research is devoted to develop computer programs capable of building models (algorithms). Many of such algorithms exist, and these often consist of various options that subtly influence performance (parameters). Furthermore, there is mathematical proof that there exists no single algorithm that works well on every dataset. This complicates the task of selecting the right algorithm for a given task.

The field of meta-learning aims to resolve these problems. The purpose is to determine what kind of algorithms work well on which datasets. This is often done experimentally. A common approach is to execute many algorithms on many datasets, and based on the results learn which combinations work well (hence the name: meta-learning). Although this works well in practise, time is a limiting factor. Many of these experiments are very time-consuming, which unintentionally limits the scale of many studies.

In order to solve this problem, we have developed OpenML. This is an online database on which researches can share experimental results amongst each other, potentially scaling up the size of meta-learning studies. Having earlier experimental results freely accessible and reusable for others, it is no longer required to conduct time expensive experiments. Rather, researchers can answer such experimental questions by a simple database look-up.

This thesis addresses how OpenML can be used to answer fundamental meta-learning questions such as 'which algorithm should be preferred on a certain kind of data?', 'what is the effect of a given parameter on the performance of an algorithm?' and 'how does a newly developed algorithm perform compared to existing algorithms?'

Curriculum Vitae

Jan van Rijn was born on Wednesday 24 June 1987 in Katwijk. From 1999 until 2004 he was a student at Andreas College (location Pieter Groen) in Katwijk. In 2008 he obtained his Bachelor of Informatics at Leiden University of Applied Sciences, after which he obtained his master Computer Science in 2012 at Leiden University (thesis: 'Playing Games: The complexity of Klondike, Mahjong, Nonograms and Animal Chess' [120]).

During his studies he participated in various Algorithm Programming Contests, among which the North Western European Regional Contest in 2010 in Bremen, Germany. He was as student-assistant actively involved in various courses of the Informatics program at Leiden University. Furthermore, during his studies he was also active as Software Developer for a small business in Amsterdam.

From 2012 until 2016 he conducted his doctoral research under the supervision of Dr. Joaquin Vanschoren, Dr. Arno J. Knobbe and Prof. Dr. Joost N. Kok at Leiden Institute of Advanced Computer Science (LIACS). During this period he played a major role in the development and maintenance of OpenML. Furthermore, he was actively involved in teaching; he assisted at various Bachelor courses and gave several guest lectures about his research.

During his time as PhD student, he made several academic visits to foreign universities. He visited the University of Waikato three times for a period of several months, as invited by Prof. Dr. Bernhard Pfahringer and Prof. Dr. Geoffrey Holmes. He also visited the University of Porto for several weeks, as invited by Prof. Dr. Pavel Brazdil. Furthermore, Jan was also a member of the Institute Council, an advisory organ within the institute, and the Social Committee, for organizing social staff events.

Currently, Jan is working as researcher at the University of Freiburg. After obtaining his PhD he wants to pursue a career in academic.

Acknowledgements

This journey started in the spring of 2012, when I was in the process of finishing my Master's Thesis. My supervisors back then, Dr. Walter Kosters and Dr. Hendrik Jan Hoogeboom, encouraged me to have an informal chat with my current advisor, Prof. Dr. Joost Kok, about the open PhD positions. It was their guidance that made me enthusiastic about scientific research and led me towards my first academic position.

Many thanks towards the active OpenML community, that made the biannual workshops something to look forward to. In particular, I would like to thank the various people that I met at several locations around the world: Bernd Bischl, Paula Branco, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Dominik Kirckhoff, Lars Kotthoff, Michel Lang, Rafael Mantovani, Luís Torgo and Joaquin Vanschoren.

I have been very lucky to work in the inspiring presence of many friends and colleagues from LIACS (and the Snellius building): Frans Birrer, Hendrik Blockeel, Benjamin van der Burgh, Ricardo Cachucho, André Deutz, Wouter Duivesteijn, Kleanthi Georgala, Vian Govers, Arno Knobbe, Rob Konijn, Michiel Kosters, Irene Martorelli, Marving Meeng, Annette Mense, Shengfa Miao, Siegfried Nijssen, Peter van der Putten, Kristian Rietveld, Jurriaan Rot, Claudio Sá, Marijn Schraagen, Hanna Schraffenberg, Marijn Swenne, Mima Stanojkovski, Anna Stawska, Frank Takes, Ugo Vespier, Jonathan Vis and Rudy van Vliet.

I would like to thank Prof. Dr. Bernhard Pfahringer and Prof. Dr. Geoffrey Holmes for hosting me multiple times at the University of Waikato. It has been a great pleasure to work together in such a prestigious and friendly environment. I also would like to thank several others from the Department of Computer Science, who made my visits both instructive and interesting: Christopher Beckham, Felipe Bravo-Marquez, Bob Durant, Dale Fletcher, Eibe Frank, Henry Gouk, Brian Hardyment, Simon Laing, Tim Leathart, Michael Mayo, Peter Reutemann, Sam Sarjant and Quan Sun. Our time in the lab was silver, but our time in the Tea Room was golden.

Furthermore, I would like to thank prof. dr. Pavel Brazdil and dr. Carlos Soares for inviting me to visit the University of Porto. It was a great experience to spend such cold months in the warm environment of LIAAD. Also many thanks to Salisu Abdulrahman, for the great collaborations prior, during and after my stay in Portugal.

Finally, I would like to mention my family and friends across the world. In particular, Nico, Leuntje, Niels, Annelies, Bas and Leoni van Rijn, for providing the warm and stable environment that I always could rely on. This thesis is dedicated to you.

Publication List

Below is a chronological list of publications by the author up to October 2016.

- S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning, Special Issue on Metalearning and Algorithm Selection*, forthcoming, 2016
- M. J. Post, P. van der Putten, and J. N. van Rijn. Does Feature Selection Improve Classification? A Large Scale Experiment in OpenML. In *Advances in Intelligent Data Analysis XV*, pages 158–170. Springer, 2016
- J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren. On the Evaluation of Algorithm Selection Problems. In *Proceedings of the 25th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2016)*, 2 pages, 2016
- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Having a Blast: Meta-Learning and Heterogeneous Ensembles for Data Streams. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 1003–1008. IEEE, 2015
- J. N. van Rijn, F. W. Takes, and J. K. Vis. The complexity of rummikub problems. In *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC 2015)*, 8 pages, 2015
- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Case Study on Bagging Stable Classifiers for Data Streams. In *Proceedings of the 24th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2015)*, 6 pages, 2015
- J. N. van Rijn and J. Vanschoren. Sharing RapidMiner Workflows and Experiments with OpenML. In J. Vanschoren, P. Brazdil, C. Giraud-Carrier, and L. Kotthoff, editors, *Proceedings of the 2015 International Workshop on Meta-Learning*

and Algorithm Selection (MetaSel), number 1455 in CEUR Workshop Proceedings, pages 93–103, Aachen, 2015

- S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren. Algorithm selection via meta-learning and sample-based active testing. In J. Vanschoren, P. Brazdil, C. Giraud-Carrier, and L. Kotthoff, editors, *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection (MetaSel)*, number 1455 in CEUR Workshop Proceedings, pages 55–66, Aachen, 2015

- J. Vanschoren, J. N. van Rijn, B. Bischl, G. Casalicchio, M. Lang, and M. Feurer. OpenML: a Networked Science Platform for Machine Learning. In *ICML 2015 MLOSS Workshop*, 3 pages, 2015

- J. Vanschoren, J. N. van Rijn, and B. Bischl. Taking machine learning research online with OpenML. In *Proceedings of the 4th International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 1–4. JLMR.org, 2015

- J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren. Fast Algorithm Selection using Learning Curves. In *Advances in Intelligent Data Analysis XIV*, pages 298–309. Springer, 2015

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

- J. N. van Rijn and J. K. Vis. Endgame Analysis of Dou Shou Qi. *ICGA Journal*, 37(2):120–124, 2014

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Towards Meta-learning over Data Streams. In J. Vanschoren, P. Brazdil, C. Soares, and L. Kotthoff, editors, *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, number 1201 in CEUR Workshop Proceedings, pages 37–38, Aachen, 2014

- J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Algorithm Selection on Data Streams. In *Discovery Science*, volume 8777 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2014

- H. J. Hoogeboom, W. A. Kosters, J. N. van Rijn, and J. K. Vis. Acyclic Constraint Logic and Games. *ICGA Journal*, 37(1):3–16, 2014

- J. N. van Rijn and J. K. Vis. Complexity and retrograde analysis of the game dou shou qi. In *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence*, 8 pages, 2013
- J. N. Van Rijn, V. Umaashankar, S. Fischer, B. Bischl, L. Torgo, B. Gao, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. A RapidMiner extension for Open Machine Learning. In *RapidMiner Community Meeting and Conference*, pages 59–70, 2013
- J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. OpenML: A Collaborative Science Platform. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–649. Springer, 2013
- J. N. Van Rijn and J. Vanschoren. OpenML: An Open Science Platform for Machine Learning. In *Proceedings of the 22th Belgian-Dutch Conference on Machine Learning (BeNeLearn 2013)*, 1 page, 2013

Titles in the IPA Dissertation Series since 2013

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of

Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty

of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security pro-*

tocols. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction*. Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things*. Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes*. Faculty of Sciences, Department of Computer Science, VUA. 2015-20

M. Zaharieva-Stojanovski. *Closer to Reliable Software: Verifying functional behaviour of concurrent programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21

R.J. Krebbers. *The C standard formalized in Coq*. Faculty of Science, Mathematics and Computer Science, RU. 2015-22

R. van Vliet. *DNA Expressions – A Formal Notation for DNA*. Faculty of Mathematics and Natural Sciences, UL. 2015-23

S.-S.T.Q. Jongmans. *Automata-Theoretic Protocol Programming*. Faculty of Mathematics and Natural Sciences, UL. 2016-01

S.J.C. Joosten. *Verification of Interconnects*. Faculty of Mathematics and Computer Science, TU/e. 2016-02

M.W. Gazda. *Fixpoint Logic, Games, and Relations of Consequence*. Faculty of Mathematics and Computer Science, TU/e. 2016-03

S. Keshishzadeh. *Formal Analysis and Verification of Embedded Systems for Healthcare*. Faculty of Mathematics and Computer Science, TU/e. 2016-04

P.M. Heck. *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05

Y. Luo. *From Conceptual Models to Safety Assurance – Applying Model-Based Tech-*

niques to Support Safety Assurance. Faculty of Mathematics and Computer Science, TU/e. 2016-06

B. Ege. *Physical Security Analysis of Embedded Devices*. Faculty of Science, Mathematics and Computer Science, RU. 2016-07

A.I. van Goethem. *Algorithms for Curved Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2016-08

T. van Dijk. *Sylvan: Multi-core Decision Diagrams*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09

I. David. *Run-time resource management for component-based systems*. Faculty of Mathematics and Computer Science, TU/e. 2016-10

A.C. van Hulst. *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs*. Faculty of Mechanical Engineering, TU/e. 2016-11

A. Zawedde. *Modeling the Dynamics of Requirements Process Improvement*. Faculty of Mathematics and Computer Science, TU/e. 2016-12

F.M.J. van den Broek. *Mobile Communication Security*. Faculty of Science, Mathematics and Computer Science, RU. 2016-13

J.N. van Rijn. *Massively Collaborative Machine Learning*. Faculty of Mathematics and Natural Sciences, UL. 2016-14