Giuseppe Casalicchio

# On Benchmark Experiments and Visualization Methods for the Evaluation and Interpretation of Machine Learning Models

Dissertation an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Eingereicht am 07.01.2019

Giuseppe Casalicchio

# On Benchmark Experiments and Visualization Methods for the Evaluation and Interpretation of Machine Learning Models

Dissertation an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Eingereicht am 07.01.2019

Erster Berichterstatter: Prof. Dr. Bernd Bischl
Zweiter Berichterstatter: Prof. Dr. Friedrich Leisch
Dritter Berichterstatter: PD Dr. Fabian Scheipl

Tag der Disputation: 18.03.2019

## Acknowledgments

*This thesis would not have been possible without the help, support, guidance, and advice of many people! In particular, I would like to express my sincere gratitude to ...*

- ... *my supervisor Prof. Dr. Bernd Bischl for the seamless collaboration, trust, support, encouragement and many advice throughout the years.*
- ... *Prof. Dr. Friedrich Leisch and PD Dr. Fabian Scheipl for their willingness to act as the second and third reviewer for my Ph.D. thesis.*
- ... *Prof. Dr. Christian Heumann and Prof. Dr. Helmut Küchenhoff for their availability to be part of the examination panel at my Ph.D. defense.*
- ... *Prof. Dr. Matthias Schmidt for the excellent supervision and guidance in my first research project.*
- ... *Prof. Dr. Helmut Küchenhoff for the financial support during my work at the StabLab, which gave me the opportunity to work on many interesting consulting projects.*
- ... *the Centre Digitisation.Bavaria (ZD.B) and the LMUexcellent program for financial support during my work on these projects.*
- ... *Prof. Dr. Joaquin Vanschoren and Jan van Rijn for pushing forward the OpenML project and organizing many delightful workshops and hackathons with lots of free lunches.*
- ... *all my coauthors for the fruitful collaborations.*
- ... *all members of my working group: Thank you for the great collaboration, inspiring discussions and unforgettable time at different workshops and conferences!*
- ... *all current and former office colleagues: Thank you Lisa Möst, Janek Thomas, Xudong Sun, Daniel Schalk, Stefan Coors for withstanding all my noises and monologues in the office.*
- ... *all remaining former and current colleagues at the Department of Statistics for the excellent general atmosphere.*
- ... *my parents and the best girlfriend who always support me, understand me (most of the time) and are always there for me.*

## Zusammenfassung

Die vorliegende kumulative Dissertation umfasst fünf wissenschaftliche Beiträge, die in drei Teilen gegliedert sind. Der erste Teil der Arbeit erweitert das mlr Paket für die statistische Software R. Das Paket bietet ein generisches, objektorientiertes und leicht erweiterbares Grundgerüst für maschinelles Lernen und die Durchführung von Benchmark-Experimenten. Der erste wissenschaftliche Beitrag ist in Kapitel 3 eingebettet und präsentiert mehrere Verfahren für Multilabel-Klassifikation, die auch im mlr Paket implementiert und in einer Benchmark-Studie miteinander verglichen wurden.

Der zweite Teil dieser Arbeit konzentriert sich auf die Vereinfachung von Benchmark-Experimenten mithilfe der Onlineplattform OpenML.org. Die Plattform ermöglicht es unter anderem Datensätze und Ergebnisse aus Benchmark-Experimenten einfach in maschinenlesbarer Form zu verwalten und Forschern und Anwendern aus aller Welt frei zugänglich zu machen. Kapitel 4 stellt das R Paket OpenML vor, welches eine einfache Schnittstelle für die Kommunikation mit dem OpenML-Server direkt aus R heraus ermöglicht und somit die Suche, sowie den Download und Upload von Datensätzen und Benchmark-Ergebnissen erleichtert. In Kapitel 5 wird für der Einsatz von Benchmarking-Suiten geworben (d.h. einer Sammlung von sorgfältig ausgewählten und leicht zugänglichen Datensätzen) und eine Möglichkeit auf der OpenML Plattform geschaffen eigene Benchmarking-Suiten zu erstellen. Weiterhin wird eine erste solche Sammlung für Klassifikationsdatensätze vorgeschlagen (die *OpenML100* Benchmarking-Suite). Die *OpenML100* Suite wurde sorgfältig aus Tausenden von Datensätzen aus OpenML zusammengestellt und stellt eine große Auswahl an gut dokumentierten Datensätzen aus verschiedenen Bereichen mit umfangreichen Metadaten zur Verfügung. Darüber hinaus werden auch die Datensplits für Resampling-Methoden bereitgestellt, die eine reproduzierbare, besser vergleichbare und standardisierte Analyse der Benchmark-Ergebnisse ermöglichen.

Der dritte Teil dieser Arbeit beschäftigt sich mit Visualisierungsmethoden für die Beurteilung und Interpretation von Vorhersagemodellen. Kapitel 6 legt den Fokus auf eine Erweiterung der sogenannten *predictiveness* Kurve, welches als visuelles Werkzeug zur Beurteilung der Leistung von Vorhersagemodellen vorgeschlagen wurde. Im Gegensatz zur *receiver operating characteristic* (ROC) Kurve, berücksichtigt die predictiveness Kurve zusätzlich auch die Kalibrierung der Vorhersagen. Im Rahmen der in Kapitel 6 vorgeschlagenen Erweiterung wird ein neuartiges visuelles Werkzeug vorgestellt, die *residual-based predictiveness* (RBP) Kurve, welches verschiedene Mängel der herkömmlichen predictiveness Kurve behebt. Kapitel 7 gibt einen Überblick über gängige modell-agnostische Interpretationsmethoden für maschinelle Lernverfahren und präsentiert dann ein Verfahren für die lokale Merkmalswichtigkeit, d.h. der Beitrag einzelner Beobachtungen auf die Merkmalswichtigkeit. Daraus werden dann zwei neuartige visuelle Werkzeuge abgeleitet, die veranschaulichen sollen, wie sich Änderungen in einem Merkmal sowohl auf die globale als auch lokale Merkmalswichtigkeit auswirken. Darüber hinaus wird ein weiteres Maß für die Merkmalswichtigkeit vorgeschlagen, welches die Gesamtperformance eines Modells fair auf die Beiträge der einzelnen Merkmale verteilt, um die Merkmalswichtigkeit auch Modell übergreifend sinnvoll vergleichen zu können.

## Summary

This cumulative dissertation consists of five contributing articles, which are divided into three parts. The first part of the thesis extends the mlr package for the statistical software R. Specifically, the package provides a generic, object-oriented, and easily extensible framework for machine learning and benchmark experiments. The first contributing article is embedded in Chapter 3 and is concerned with the implementation of several multilabel classification methods into the mlr package. These methods are first described and then, after being implemented into the mlr package, compared in a benchmark study.

The second part of this work focuses on the simplification of benchmark experiments using the online machine learning platform OpenML.org. One of the capabilities of the platform is to organize datasets and results of benchmark experiments online in a machine-readable form, thereby making them freely accessible to researchers from all over the world. The second contribution included in Chapter 4 introduces the R package OpenML. The package provides a simple interface to communicate with the OpenML server directly from within R. Furthermore, it facilitates searching, downloading, and uploading datasets and benchmark results. The third contributing article integrated in Chapter 5 advocates the use of benchmarking suites (i.e., a collection of carefully selected and easily accessible datasets). The article proposes the *OpenML100* benchmarking suite as a first such collection of classification datasets, and it introduces an extension of the OpenML platform that allows researchers to create their own benchmarking suites. The *OpenML100* has been carefully compiled from thousands of datasets from OpenML, and it provides a wide range of well-documented datasets from various domains, together with rich meta-data. Furthermore, the *OpenML100* also includes the data splits required by resampling methods, thereby allowing for a more easily reproducible, better comparable, and standardized analysis of benchmark results.

The third part of this thesis deals with visualization methods for the evaluation and interpretation of prediction models. The fourth contributing article is concerned with an extension of the *predictiveness curve*, which is a visual tool to assess the performance of prediction models. In contrast to the *receiver operating characteristic* (ROC) curve, the predictiveness curve also considers the calibration of predictions in addition to the discrimination performance. The proposed extension in Chapter 6 describes various shortcomings of the predictiveness curve and introduces a novel visual tool to remedy most of these shortcomings, namely the *residual-based predictiveness* (RBP) curve. The last contribution in Chapter 7 gives an overview of common model-agnostic interpretability methods in machine learning and then introduces a local feature importance measure. Based on this, two novel visual tools are derived. Both tools visualize how changes in a feature affect the model performance on average, as well as for individual observations. Furthermore, the Shapley feature importance (SFIMP) measure is presented, which fairly distributes the overall model performance among the features. Thus, the SFIMP measure can also be used to compare the feature importance across different models.

# Contents

# 1. Introduction

## 1.1. Outline

This thesis focuses on the development of tools and methods that facilitate the application, comparison, as well as evaluation and interpretation of supervised machine learning methods. The next section gives an overview of all contributing articles and describes their respective goals on a general level. Furthermore, the section presents the motivation of the contributing articles, and it establishes a connection between the articles from a general perspective.

Chapter 2 introduces some basic concepts that help with understanding the broader context of this thesis, and it outlines previous and related work. The contents of the chapter should be understood as a guideline for the reader regarding the main topics of this thesis. In particular, this chapter also points out how the individual contributions of this thesis fit within the context of the topics. It should be noted that the notation used in Chapter 2 has been unified and therefore does not always match the notation used in the contributing articles embedded in Chapters 3 to 7. However, the analogies in the notation should be clear from the context.

The rest of this thesis is organized into three main parts (i.e., Part I - III). The contributing articles of this thesis are embedded within these parts as chapters (i.e., Chapter 3 - 7). At the beginning of each of these chapters, the full reference to the original publication is given, including a description of the author's specific contributions. If applicable, other information such as supplementary materials, accompanying software, and copyright information of the articles are also included. The thesis concludes with Part IV by emphasizing possible future and ongoing work.

## 1.2. Motivation and Scope

The world we live in presents us with a lot of information and phenomena, which humanity has always tried to understand better. Data is gathered by measuring and observing such information and phenomena. Hence, data contains valuable insights into the relationships in the real world that humanity aims at understanding. For this reason, companies, governments, and scientists have an ever-increasing interest in collecting data from many sources. There is the hope that the collected data may provide answers to domain-specific problems and questions in business and science applications. However, the relationships in the real world and consequently also those in collected data can be very complicated. Because of this, answers to relevant questions may remain hidden behind this complexity if data is not reported and analyzed correctly. One of the key aims in statistics and machine learning is to simplify this complexity by building a mathematical model based on the observed data. The model itself is regarded as a simplification of the real world, which is due to two main reasons. First, the data used to build the model is often limited, meaning

that several aspects of the real world remain unobserved. Second, models are only approximations with simplifying assumptions that are based on the available data.

In this dissertation, the focus is on supervised machine learning. This includes all problems that involve learning the relationship between data and an associated target outcome, with the aim to predict the outcome for new data. Such prediction tasks often occur in applications related to decision making. Whenever possible, people usually consider additional information from observed data when they try to make the best decision. Machine learning algorithms can help in this regard as they are able to learn complex relationships from historical data and produce accurate predictions for new data (Fernández-Delgado et al., 2014). Accurate predictions provide some insight into the future and can thus assist in anticipating future situations. Such insights may be used to accelerate and facilitate the decision-making process in various applications. Therefore, it is not surprising that the use of machine learning techniques is gaining more attraction in many different disciplines such as medicine (e.g., to support decisions on health care plans for patients with serious diseases) (Holmberg and Vickers, 2013; Obermeyer and Emanuel, 2016), politics (Hill and Jones, 2014), criminology (Berk et al., 2009; Wang et al., 2013), ecology (Cutler et al., 2007), and astrophysics (VanderPlas et al., 2012).

The first step in many data-driven applications is to translate an underlying domain-specific problem into a machine learning task. After this, many practitioners may still be faced with several hurdles if they consider applying machine learning algorithms to solve their domain-specific problem. This often begins with the choice of an appropriate programming language or software tool that allows for building machine learning models in a technical sense. On the one hand, such a choice depends on personal preferences, but it also depends on the quality and availability of the software tool. In the academic community, the statistical programming language R (R Core Team, 2018) is often used. One of the reasons is that R is open source and offers many excellent add-on packages for processing data (Wickham, 2017), visualizing data (Wickham, 2009), and for building predictive models (cf. Hothorn, 2018). Furthermore, it provides many other convenient functionalities and packages, including packages for dynamic report generation and reproducible research (Leisch, 2002; Xie, 2014; Stodden et al., 2014). However, the vast amount of R packages that provide algorithms for building predictive models do not always offer a unified interface. Furthermore, there is no guarantee that the corresponding output of the function calls of individual algorithms from different packages have the same structure. Such inconsistencies are due to different programming styles and practices used by the authors of the R packages, and make it more complicated and time-consuming for other researchers to write generic code for conducting and analyzing more complex machine learning experiments. As a consequence, researchers often have to write additional lines of code to unify the function calls and their corresponding outputs. Our open source R package called mlr (Bischl et al., 2016) addresses this issue by providing a modularized and unified framework for machine learning in R. In the context of this thesis, we extended the mlr package in Chapter 3 (cf. Probst et al., 2017) such that standard classification algorithms can handle multilabel classification tasks. In this way, a user benefits from all the other functionalities of the mlr package when working with multilabel classification tasks. This includes building predictive models, making predictions, assessing the algorithms' performance through different resampling techniques and different performance measures, as well as applying different methods for hyperparameter tuning and feature selection.

After deciding on a software tool that provides access to a wide range of machine learning algorithms, practitioners still need to find an appropriate subset of these algorithms, which can

also solve the underlying machine learning task as best as possible. For this purpose, it is often required to conduct *benchmark experiments* to compare the algorithms' performances. This involves assessing the performance of algorithms, which again requires two significant aspects to be considered. The first aspect is the choice of a performance measure that is best suited to assess the algorithms' performance for the machine learning task at hand. As not all performance measures are equally suitable for all types of machine learning tasks, such a choice requires both domain knowledge and knowledge about the properties of the performance measures under consideration. Section 2.2 describes this point in more detail. The second aspect is related to estimating the generalization performance of algorithms, which requires estimation methods that use the available data (repeatedly) to obtain accurate and reliable performance estimates. This topic is addressed in Section 2.3.

The selection of suitable algorithms and their comparison to find the best among all competing ones often requires an additional human expert guiding this process. However, such an expert is not always available. Furthermore, according to van Someren (2001), even experts "have often been observed to have a favorite method and to be able to transform a wide range of problems into a form that allows the method to be applied." This means that the choice of suitable algorithms is often limited to the personal experience or knowledge of the human expert and may be biased due to personal preferences. In this context, the field of meta-learning offers a data-driven alternative. One of the goals in meta-learning is to predict which algorithms are more appropriate for a given machine learning task based on previous results of benchmark experiments (Vilalta and Drissi, 2002). However, producing reliable predictions requires as many benchmark results as possible so that enough information can be provided on how different algorithms have already performed in different scenarios. The online machine learning platform OpenML (Vanschoren et al., 2013) offers access to a variety of such benchmark results, which have been shared by many other researchers. I believe that sharing and analyzing such results on a large scale (e.g., using platforms such as OpenML) provides more detailed insights into the behavior of different algorithms in different scenarios, thereby also supporting the research in the field of meta-learning. One of the long-term goals of the contributions in Part II of this thesis is to motivate other researchers to use OpenML and share their benchmark results. For this purpose, the contributions in Part II facilitate two important and rather technical aspects regarding benchmark experiments on OpenML. First, the R package OpenML is introduced in Chapter 4 (cf. Casalicchio et al., 2017). The package offers the possibility to interact with the OpenML server directly through R, thereby simplifying the work with the OpenML platform. Second, the contribution in Chapter 5 (cf. Bischl et al., 2017) promotes creating and using standardized and well-documented benchmarking suites (i.e., a collection of datasets). It also proposes a curated collection of classification datasets as a first such benchmarking suite. Here, one of the aims is to make it easier for researchers to reuse or extend existing benchmarking suites for their benchmark experiments, without having to search for appropriate datasets manually.

Another essential aspect besides looking for well-performing machine learning models is the interpretability of a model. This interpretability often refers to the ability to explain why a model produced a specific prediction. This task has attracted much attention in recent years as many machine learning models have been considered to be black boxes (Lipton, 2016; Krause et al., 2016; Guidotti et al., 2018). The contributions in Chapters 6 (cf. Casalicchio et al., 2016) and 7 (cf. Casalicchio et al., 2018) address the issue of model interpretability by developing visual tools for the evaluation of the model performance as well as the feature importance.

# 2. Methodological and General Background

## 2.1. Supervised Machine Learning

### 2.1.1. Preliminaries

In the context of supervised machine learning, it is assumed that there is an unknown functional relationship $f$ between a $p-$dimensional feature space $\mathcal{X}$ of arbitrary measurement scales and a target space $\mathcal{Y}$. Let $X = (X_1, \ldots, X_p)$ denote the corresponding random variables generated from the feature space, and let $Y$ denote the random variable generated from the target space. Supervised machine learning algorithms aim at learning the unknown functional relationship based on observed training data $\mathcal{D}_{\text{train}}$. Each observation $(\mathbf{x}, y) \in \mathcal{D}_{\text{train}}$ is drawn i.i.d. from an unknown probability distribution $\mathcal{P}$ on the joint space $\mathcal{X} \times \mathcal{Y}$. Let the vector $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)})^\top \in \mathcal{X}$ denote the feature values of the $i$-th observation. Its associated target value is denoted by $y^{(i)} \in \mathcal{Y}$. The learning algorithm uses $\mathcal{D}_{\text{train}}$ as input and induces a prediction model $\hat{f}$, which approximates $f$. This learning process is often achieved by minimizing the empirical risk $\mathcal{R}_{\text{emp}}(f)$ based on training data, i.e.,

$$\hat{f} \;=\; \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) \;=\; \arg\min_{f \in \mathcal{H}} \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{train}}} L(f(\mathbf{x}), y). \tag{2.1}$$

Here, $\mathcal{H}$ is the hypothesis space that refers to the set of all possible candidate models, and $L$ is a loss function that measures to what extent the actual target values are in concordance with the predictions generated from such a candidate model (see also Section 2.2 for some example loss functions). In particular, this means that Equation (2.1) tries to find a prediction model $\hat{f}$, for which averaging the loss function across all observations in $\mathcal{D}_{\text{train}}$ takes its minimum value.

Theoretically, Equation (2.1) can be directly minimized by finding a prediction function $\hat{f}$ for which $\hat{f}(\mathbf{x}) = y,\ \forall (\mathbf{x}, y) \in \mathcal{D}_{\text{train}}$. However, this does not necessarily result in a well-performing prediction model that also generalizes for unseen observations $(\mathbf{x}, y) \notin \mathcal{D}_{\text{train}}$, although such a prediction model is usually desired. To illustrate this issue, consider the data points in Figure 2.1 that follow a true functional relationship $f$ and randomly scatter around this underlying latent function $f$ with some noise. Furthermore, suppose a prediction model $\hat{f}$ (here, the higher degree polynomial) that only sees a subset of these data points (i.e., the training data) and is only fitted on this training data. Figure 2.1 shows that the polynomial $\hat{f}$ is sufficiently flexible to pass through every single point of the training data, thereby minimizing the empirical risk from Equation (2.1). However, the polynomial $\hat{f}$ does not appear to be a good approximation of $f$ and is said to overfit the training data since it does not generalize well. This is illustrated in Figure 2.1 by the large residuals of the polynomial for other unobserved data points. This means that the distance between the polynomial and other unobserved data points that also follow the true functional relationship $f$ is large.

To address the overfitting issue, several approaches regularize the complexity (i.e., the flexibility) of the prediction model $\hat{f}$ by minimizing Equation (2.1) with an additional penalty term $J(f)$ that measures the complexity of $\hat{f}$, i.e.,

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) \; + \; \lambda \cdot J(f). \tag{2.2}$$

Regularization usually introduces an additional parameter $\lambda$ that controls the degree of penalization. Typically, $\lambda$ is estimated by means of cross-validation. A more in-depth introduction to this topic can be found in Murphy (2013).

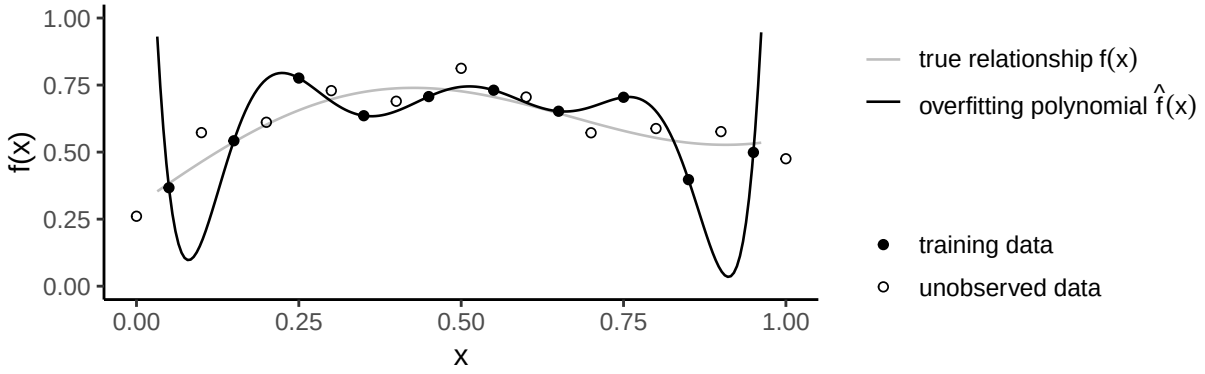



Figure 2.1.: Illustration of the overfitting issue. The polynomial $\hat{f}$ is fitted based on the training data and therefore passes through every single training data point. However, it does not appear to be a good approximation of the true functional relationship $f$ due to the large distance between $\hat{f}$ and the unobserved data points.

### 2.1.2. Learning Tasks and Algorithms

The target space $\mathcal{Y}$ typically determines the type of the machine learning task at hand. The two most fundamental learning tasks are regression and classification tasks, and classification tasks are further categorized into binary classification and multiclass classification tasks. In regression tasks, the target space is continuous, e.g., $\mathcal{Y} = \mathbb{R}$. Consequently, regression algorithms aim at predicting the continuous target $y$ through $\hat{f}(x) \in \mathbb{R}$ as accurately as possible.

Binary classification tasks have a target space that consists of two classes (e.g., $\mathcal{Y} = \{0, 1\}$). At this point, there is a further distinction between discrete (or hard) classifiers, probabilistic classifiers, and scoring (or ranking) classifiers, each producing a different type of prediction. The three different prediction types are now briefly described, since they are essential for identifying appropriate performance measures (cf. Section 2.2.2). Discrete classifiers classify observations into only one of the available classes (i.e., they directly predict discrete classes). By contrast, a probabilistic classifier aims at estimating the whole probability distribution over all classes instead of merely predicting the class membership. A scoring classifier predicts real-valued scores rather than probabilities. However, since scores are difficult to interpret, several strategies have been developed to transform (i.e., calibrate) these scores to obtain values that can be interpreted as posterior probabilities (see also Section 2.2.2). From this point on, the predictions of discrete classifiers are denoted as $\hat{h}(x) \in \{0, 1\}$, while predictions of probabilistic classifiers are denoted

as $\hat{\pi}(x) \in [0, 1]$. As in the regression scenario, the notation of continuous predictions of scoring classifiers is kept as $\hat{f}(x) \in \mathbb{R}$.

Another type of machine learning tasks are multiclass classification tasks, which can be viewed as a generalization of binary classification tasks where the target space is a finite set with at least three discrete classes, e.g., $\mathcal{Y} = \{1, \ldots, m\}$ with $m \geq 3$. Here, the aim is to classify an observation into only one of the $m$ available classes. Due to the similarity between binary and multiclass classification tasks, several classifiers naturally also account for the multiclass case, such as decision trees. However, there are also approaches that allow transforming multiclass classification tasks into multiple binary classification tasks, such as the one-versus-rest and the one-versus-one approach (cf. Bishop, 2006, Ch. 4.1.2). Based on such approaches, it is possible to use any binary classifier to solve multiclass classification tasks.

There are also more complex tasks such as multilabel classification. Here, the binary target space is of finite dimensionality $m$, e.g., $\mathcal{Y} = \{0, 1\}^m$, where $m \geq 2$. The target outcome is denoted by the vector $\mathbf{y}^{(i)} = (y_1^{(i)}, \ldots, y_m^{(i)})^\top \in \mathcal{Y}$. The aim is to classify an observation in up to $m$ different classes at the same time, i.e., without limiting the number of classes to which the observation is classified. This aim is in contrast to multiclass classification where observations can only be classified to one single class. The most straightforward approach to address multilabel classification is to consider each target value separately and fit $m$ binary classifiers, specifically one for each target value. However, there are other more sophisticated adaption and transformation methods based on binary classifiers that have been proposed in the literature for approaching multilabel classification tasks. Chapter 3 discusses and compares a few of these methods, and it also provides an implementation of several transformation methods into the mlr package.

## 2.2. Performance Measures

Before practitioners can evaluate the performance of machine learning algorithms, they first have to consider two significant aspects. First, they need to choose a performance measure that is best suited for the machine learning task at hand and reasonable for the pursued goal of the underlying application. As not all performance measures are equally suitable for all types of machine learning tasks, such a decision requires both domain knowledge and knowledge about the properties of the performance measures under consideration. Second, practitioners need to use an estimation technique that can estimate the generalization performance of the algorithm based on the available data as accurately as possible. This section focuses on the former aspect and summarizes a few common performance measures that are essential for evaluating the performance of machine learning models. For the sake of simplicity, the section describes how the performance of such models can be estimated based on a set of observations that were not used to fit the prediction model. This set is referred to as the test set $\mathcal{D}_{\text{test}} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$, where $n = |\mathcal{D}_{\text{test}}|$. Section 2.3 then describes in more detail the second aspect of how the available data can be efficiently used to obtain a reliable estimate of the generalization performance.

Performance measures are quantitative values that reflect how well the predictions of a machine learning model match the ground truth (i.e., the actual target values). In particular, many performance measures can be expressed in terms of a loss function.[1] It should be noted that the

[1] In the case where higher performance values signify a better performance, the loss function is often included in the definition of the performance measure with a negative sign (e.g., see the minus sign in Equation (2.4)).

loss function included in the definition of a performance measure does not necessarily have to coincide with the loss function that occurs in Equation (2.1) when minimizing the empirical risk. Although such a concordance is preferable in many applications, it is not always possible. The reason for this is that the former loss function is typically used to assess the performance of a machine learning algorithm *after* the learning process. It allows practitioners to consider multiple performance measures simultaneously, which may also be of interest to them depending on the application. By contrast, the latter loss function defines the objective function, which is directly minimized by the machine learning algorithm *during* the learning process. In particular, this loss function cannot always be chosen arbitrarily for at least two reasons. First, many algorithms already implicitly minimize a specific loss function that cannot be replaced due to the design of the algorithm. Second, some algorithms allow the use of different loss functions, but they still require a loss function that satisfies several requirements (e.g., some algorithms require differentiable loss functions).

There are usually many performance measures, and the development of new ones is undoubtedly possible, each coming with its strengths and weaknesses. Consequently, the performance measures described in the following sections should by no means be regarded as exhaustive. Most notably, the focus in this section is more on performance measures for binary classification tasks, since they serve as a basis for the performance assessment in Chapter 3 in the context of multilabel classification and play a significant role in the visual performance assessment in Chapter 6. Furthermore, there are also several types of visual performance measures. While quantitative performance measures are scalar values reflecting the overall performance, visual performance measures aim at visualizing more details that often remain hidden in scalar values.

### 2.2.1. Measures for Regression

Many performance measures for regression tasks are based on the estimated model residuals $\hat{\epsilon} = y - \hat{f}(\mathbf{x})$, which refer to the difference between the actual target value and the associated prediction of the model. The two most widely used loss functions in such situations are the $L_1$-loss and the $L_2$-loss.[2] Both can be written in terms of the model residuals, i.e.,

$$L_1(\hat{f}(\mathbf{x}), y) = |y - \hat{f}(\mathbf{x})| = |\hat{\epsilon}| \quad \text{and} \quad L_2(\hat{f}(\mathbf{x}), y) = (y - \hat{f}(\mathbf{x}))^2 = \hat{\epsilon}^2.$$

In regression settings, the mean squared error ($MSE$) is frequently used to measure the performance. It is estimated by averaging the $L_2$-loss across all observations involved in the computation of the $MSE$, i.e.,

$$\widehat{MSE} = \frac{1}{n} \sum_{i=1}^{n} L_2(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{f}(\mathbf{x}^{(i)}))^2 = \frac{1}{n} \sum_{i=1}^{n} (\hat{\epsilon}^{(i)})^2.$$

Since the square is being taken for each term, the $MSE$ puts more weight on outlier predictions in the case where $|\hat{\epsilon}^{(i)}| > 1$. If predictions are far from their associated actual target values, their model residuals $\hat{\epsilon}^{(i)}$ will be large, and squaring them will make their contribution to the MSE even

[2] Note that using the $L_1$-loss in the minimization problem from Equation (2.1) leads to predictions that estimate the conditional median, while using the $L_2$-loss leads to predictions that estimate the conditional mean of the distribution of $Y$ given $X$ (cf. Cramer (1946, Ch. 15) and Shynk (2012, pp. 545 – 553)).

larger. If this property is not desirable, it is possible to use the mean absolute error ($MAE$) as an outlier-robust alternative, which is based on the $L_1$-loss, i.e.,

$$\widehat{MAE} = \frac{1}{n}\sum_{i=1}^{n} L_1(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}) = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{f}(\mathbf{x}^{(i)})| = \frac{1}{n}\sum_{i=1}^{n} |\hat{\epsilon}^{(i)}|.$$

Furthermore, many transformations and modifications of the $MSE$ are also frequently used. For example, the root mean squared error ($RMSE$) or the root mean squared logarithmic error ($RMSLE$). The former provides better interpretable performance values as they are on the same scale as the target values. The latter is often used when the target values are non-negative, such as in the case of count data.

### 2.2.2. Measures for Binary Classification

As mentioned, discrete classifiers, scoring classifiers, and probabilistic classifiers produce different predictions types. Figure 2.2 illustrates how these different types of predictions (i.e., discrete classes, scores, and probabilities) can be converted into another prediction type. There are specialized performance measures for each of these three prediction types, and some of these measures are described in the following paragraphs. The easiest way to assess the performance is to directly use an appropriate performance measure for the corresponding prediction type produced by the classifier. Alternatively, the predictions can be converted into a different type as illustrated in Figure 2.2. This allows practitioners to also consider other performance measures, which are more appropriate for the converted predictions.

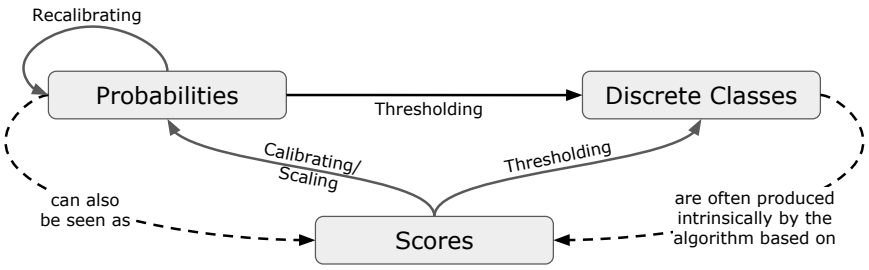



Figure 2.2.: Illustration of how different types of predictions can be converted into another type. The topics of recalibrating, calibrating, and thresholding are also referred to in the corresponding paragraphs below.

#### Discrete Classifiers

The two discrete classes in binary classification are often referred to as the *positive* class and the *negative* class.[3] As the aim of discrete classifiers is to predict discrete classes, many performance measures are mainly concerned with measuring the *discrimination performance*, i.e., the ability of a classifier to separate those classes. The confusion matrix is a $2 \times 2$ contingency table that can help in this respect. It summarizes all four possible outcomes that can occur when the predictions

[3]This terminology has its origin in a different notation of the target space, in which the class labels are $-1$ or $+1$. However, the notation here denotes the classes as 0 or 1. Thus, the *positive* class refers to class 1, and the *negative* class refers to class 0.

$\hat{h}(\mathbf{x})$ of a discrete classifier are compared with the ground truth, i.e., the associated actual classes $y$. More specifically, the confusion matrix in Table 2.1 reports the number of true negatives ($TN$), true positives ($TP$), false negatives ($FN$) and false positives ($FP$).

| | | Ground Truth | |
|---|---|---|---|
| | | Positive $Y = 1$ | Negative $Y = 0$ |
| Prediction | Positive $\hat{h}(X) = 1$ | $TP$ | $FP$ |
| | Negative $\hat{h}(X) = 0$ | $FN$ | $TN$ |
| | Total | $TP + FN$ | $FP + TN$ |

Table 2.1.: An illustration of a $2\times2$ confusion matrix. $TN$ and $TP$ refer to the number of predicted classes that were correctly classified as the negative and positive class, respectively. $FN$ and $FP$ refer to the positive and negative classes of the ground truth that were wrongly classified as the negative and positive class, respectively.

Various performance measures can be derived from the confusion matrix (cf. Sokolova et al., 2006; Sokolova and Lapalme, 2009). Among them is the classification accuracy ($ACC$), which corresponds to the proportion of correctly classified observations. The $ACC$ can be calculated from the elements of the confusion matrix by $\frac{TP+TN}{TP+FN+FP+TN}$. Many performance measures that are based on the confusion matrix can be expressed in terms of the zero-one loss (also referred to as the indicator loss) function, which is defined as

$$L_{0/1}(\hat{h}(\mathbf{x}), y) = I(\hat{h}(\mathbf{x}) \neq y) = \begin{cases} 1 & \text{if } \hat{h}(\mathbf{x}) \neq y \\ 0 & \text{if } \hat{h}(\mathbf{x}) = y \end{cases}. \tag{2.3}$$

For the aforementioned classification accuracy, this can be best illustrated by first writing the $ACC$ in terms of a probability, i.e.,

$$ACC = \mathbb{P}(\hat{h}(X) = Y) = 1 - \mathbb{P}(\hat{h}(X) \neq Y).$$

The $ACC$ is then estimated using the set of observations $\mathcal{D}_{\text{test}} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ and the zero-one loss from Equation (2.3) by

$$\widehat{ACC} = 1 - \frac{1}{n}\sum_{i=1}^{n} L_{0/1}(\hat{h}(\mathbf{x}^{(i)}), y^{(i)}). \tag{2.4}$$

Two other commonly used measures are the true positive rate ($TPR$) and the false positive rate ($FPR$). The $TPR$ is defined as the proportion of correctly classified observations among the ones with positive actual class, which can be expressed as $\frac{TP}{TP+FN}$. It can also be written as a conditional probability, i.e.,

$$TPR = \mathbb{P}(\hat{h}(X) = Y | Y = 1) = 1 - \mathbb{P}(\hat{h}(X) \neq Y | Y = 1).$$

The $FPR$ refers to the proportion of wrongly classified observations among the ones with negative actual class (i.e., $\frac{FP}{FP+TN}$), and it can be written as

$$FPR = \mathbb{P}(\hat{h}(X) \neq Y | Y = 0).$$

Both $TPR$ and $FPR$ can therefore be estimated using the zero-one loss, i.e.,

$$\widehat{TPR} = 1 - \frac{1}{n_1} \sum_{i:\, y^{(i)}=1} L_{0/1}(\hat{h}(\mathbf{x}^{(i)}), y^{(i)}) \quad \text{and} \quad \widehat{FPR} = \frac{1}{n_0} \sum_{i:\, y^{(i)}=0} L_{0/1}(\hat{h}(\mathbf{x}^{(i)}), y^{(i)}), \tag{2.5}$$

where $n_0 = \sum_{i=1}^{n} I(y^{(i)} = 0)$ and $n_1 = \sum_{i=1}^{n} I(y^{(i)} = 1)$ denote the number of observations within each class from the ground truth. In general, there is a trade-off between the $TPR$ and $FPR$. In fact, always predicting the positive class results in a $TPR$ of 1 (i.e., the best possible value), which is because the positive class is implicitly always correctly classified. At the same time, the negative class is always wrongly classified, yielding a $FPR$ of 1 (i.e., the worst possible value for the $FPR$). Figure 2.3, panel (a) illustrates how this trade-off can be visualized in the receiver operating characteristic (ROC) space, which is a 2-dimensional plot that depicts $FPR$ vs. $TPR$ (Fawcett, 2004). Each discrete classifier yields a single point in the ROC space. For example, the best possible classifier has a $FPR$ of 0 and $TPR$ of 1, and in such a case the point is located in the upper left corner of the ROC space. A random guessing classifier that predicts the positive class (i.e., class 1) at random with some constant probability is always located on the identity line (i.e., the baseline).

Two further aspects should be considered in the context of discrete classes. The first aspect refers to situations where the classes are highly imbalanced. For example, using the $ACC$ in such situations is not appropriate and can result in misleading interpretations of the performance. The reason for this is best illustrated by a model that always predicts the majority class and consequently misclassifies all observations belonging to the minority class. In this case, the $ACC$ value would still be high, indicating a well-functioning model, although observations from the minority class are always misclassified. Thus, it is preferable to use performance measures that also take the class imbalance into account, such as the balanced accuracy $BACC = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{FP+TN}\right)$ (Brodersen et al., 2010). The $BACC$ offers a better alternative in the case of imbalanced classes, since it is defined as the arithmetic mean of the classification errors in both classes, and thus it weights both errors equally. Another alternative is the F-measure, which is the harmonic mean between the $TPR$ (also known as recall) and the precision measure (i.e., $Precision = \frac{TP}{TP+FP}$) (Hripcsak and Rothschild, 2005). The second aspect is concerned with applications that involve different (i.e., asymmetric) costs for specific parts of the confusion matrix in Table 2.1. For example, in medical diagnosis, the cost of FP (i.e., classifying a healthy patient as diseased) is often different from the cost of FN (i.e., classifying a diseased patient as healthy). Here, it is reasonable to use a modified version of the balanced accuracy by using weights that are proportional to the costs instead of using equal weights. Such a *weighted accuracy* with varying weights was proposed by Androutsopoulos et al. (2000). Similarly, other measures that also take costs into account could be used (cf. Hidalgo, 2002).

At this point, it should be noted that many discrete classifiers are inherently scoring or probabilistic classifiers, although they directly predict class memberships (see Figure 2.2). This is because they often internally compute scores or probabilities and determine the class membership based on these values (cf. Fawcett, 2004).

### Scoring Classifiers

Scoring classifiers predict scores that usually range from $-\infty$ to $\infty$, i.e., $\hat{f}(\mathbf{x}) \in \mathbb{R}$. In particular, different scoring classifiers may predict scores that are on a completely different scale. Consequently, it is often only the order of the scores that is relevant when comparing different scoring classifiers. Therefore, the scale and magnitude of the scores can often be neglected. The easiest way to compare and assess the performance of scoring classifiers is to transform them into discrete classifiers (see Figure 2.2). This allows the use of previously described discrimination measures, which are based on the confusion matrix. For this purpose, a threshold $\theta$ is usually used to separate the scores into discrete classes. The resulting discrete classifier for a given threshold $\theta$ is then

$$\hat{h}(\mathbf{x}, \theta) = \begin{cases} 1 & \text{if } \hat{f}(\mathbf{x}) > \theta \\ 0 & \text{if } \hat{f}(\mathbf{x}) \leq \theta \end{cases}. \tag{2.6}$$

However, any value within the range of the predicted scores can act as a threshold, and different thresholds might result in an entirely different confusion matrix. Therefore, it is not trivial how to assess the performance in such situations and how to select an appropriate threshold beforehand. The most common way is to select the threshold value either manually (e.g., a pre-specified threshold value of interest) or based on optimizing specific criteria (cf. Fluss et al., 2005; Perkins and Schisterman, 2006).

The ROC curve has proven to be useful for assessing the performance of a scoring classifier because it visualizes the resulting $TPR(\theta)$ and $FPR(\theta)$ for each possible threshold $\theta$. Thus, it implicitly considers the order of the scores imposed by the scoring classifier (cf. Hernández-Orallo et al., 2012). Specifically, the $TPR$ and $FPR$ at a given threshold value $\theta$ are defined respectively as

$$TPR(\theta) = \mathbb{P}(\hat{f}(X) > \theta | Y = 1) \quad \text{and} \quad FPR(\theta) = \mathbb{P}(\hat{f}(X) > \theta | Y = 0).$$

The pairs $(\widehat{FPR}(\theta), \widehat{TPR}(\theta))$ are computed using Equation (2.5) based on different threshold values $\theta$ and by using $\hat{h}(\mathbf{x}, \theta)$ from Equation (2.6). The points can then be visualized in the ROC space, yielding an estimate of the ROC curve when connecting these points.

A natural performance measure that can be derived from the ROC curve is the area under the curve ($AUC$), which according to Pencina et al. (2008) can be defined as

$$AUC = \int_0^1 TPR(\theta) \, \frac{\mathrm{d}}{\mathrm{d}\theta} \, FPR(\theta) \, \mathrm{d}\theta.$$

Obviously, the $AUC$ integrates over all possible thresholds $\theta$ across the whole range of scores. For illustration purposes, Figure 2.3, panel (b) displays an optimal ROC curve with $AUC = 1$ and a ROC curve with an $AUC = 0.8$. The dashed identity line refers to the ROC curve of a random guessing classifier and is used as the baseline. Since the area of the baseline is equal to 0.5, the estimated $AUC$ of a random guessing classifier will also be close to 0.5.

### Probabilistic Classifiers

Probabilistic classifiers are often preferred in applications where it is desirable to report probabilities instead of discrete classes or real-valued scores. For instance, in the context of biomedical research, probabilistic classifiers can be very useful as they allow to quantify the risk of having a

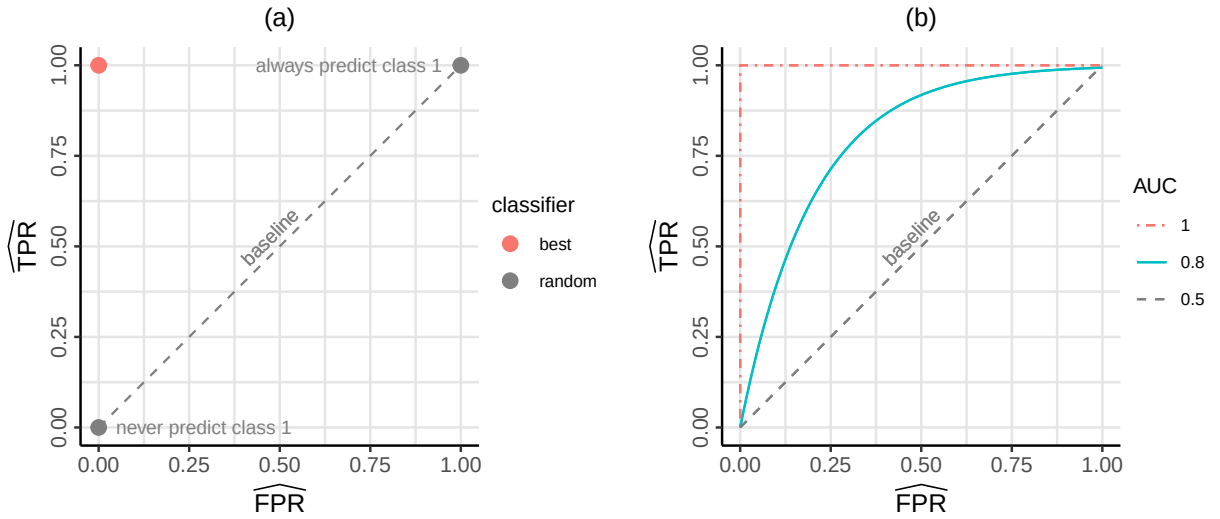



Figure 2.3.: Panel (a) displays the best possible discrete classifier in the ROC space and two random guessing classifiers lying on the baseline, namely one that always predicts class 1 and one that never predicts class 1. Panel (b) displays the optimal ROC curve (with $AUC = 1$), a ROC curve with $AUC = 0.8$, and the ROC curve of a random guessing classifier (with $AUC = 0.5$).

disease. Probabilities can be considered as scores lying between 0 and 1. Thus, it is possible to use the same discrimination measures as those for scoring and discrete classifiers to assess the performance of probabilistic classifiers. However, another essential criterion for assessing probabilistic classifiers besides the use of discrimination measures is the calibration of the predicted probabilities. For well-calibrated predictions, it is typically expected that the predicted probabilities of an event are, on average, close to the relative frequency for which that event actually occurred within the ground truth. In contrast to scoring classifiers, the magnitude of the predicted probabilities is therefore relevant when assessing the performance of probabilistic classifiers.

For this reason, more appropriate performance measures such as the log-loss (cf. Buja et al., 2005) or the Brier score ($BS$) (Brier, 1950) have been proposed for probabilistic classifiers. These measures also take into account the magnitude of the predicted probabilities. For example, the Brier score, which corresponds to the $MSE$ of the predicted probabilities $\hat{\pi}(\mathbf{x})$, is defined as

$$\widehat{BS} = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{\pi}(\mathbf{x}^{(i)}))^2.$$

In particular, the log-loss measure as well as the Brier score are known to take into account both discrimination and calibration performance. This is because both measures belong to the class of strictly proper scoring rules (Gneiting and Raftery, 2007; Steyerberg et al., 2010). In fact, Murphy (1972, 1973) showed that the Brier score is decomposable in at least two components, that is, a measure of calibration (also referred to as reliability) and a measure of discrimination (also referred to as refinement loss). According to Hernández-Orallo et al. (2012), the decomposition of the Brier score also "suggests a connection between the Brier score and ROC curves, and particularly between refinement loss and $AUC$, since both are performance metrics which do not require the magnitude of the scores of the model." The link between the discrimination component

of the Brier score (i.e., the refinement loss) and the $AUC$ emphasizes once again that the $AUC$ is only a discrimination measure and does not assess the calibration of predicted probabilities.

As mentioned in the previous paragraph, it is reasonable to assess the performance of scoring classifiers by utilizing ROC curves. However, the use of a ROC curve and its AUC for assessing the performance of probabilistic classifiers has often been criticized because these measures only take into account the discrimination performance (Hanley and McNeil, 1982; Cook, 2007). Furthermore, the ROC curve does not directly visualize either the magnitude of the predicted probabilities or the magnitude of the threshold values (Pepe et al., 2008). Therefore, many other visualization methods have been proposed to remedy these deficiencies, such as the Brier curve (Hernández-Orallo et al., 2011) or the predictiveness curve (Huang et al., 2007; Pepe et al., 2008). The predictiveness curve and several of its shortcomings are described in Chapter 6 in more detail. The contribution in Chapter 6 introduces the residual-based predictiveness (RBP) curve, which extends the original predictiveness curve and addresses several of its shortcomings. Furthermore, the chapter also relates the RBP curve to several other measures, such as the $TPR$ and $FPR$ at given threshold values (i.e., discrimination measures), the Hosmer-Lemeshow statistic (i.e., a calibration measure also known as the *calibration across deciles*) (Hosmer and Lemesbow, 1980; Lemeshow and Hosmer, 1982), and the $MAE$.

It is possible to convert the scores predicted by scoring classifiers into posterior probabilities by using methods, such as Platt scaling (Platt et al., 1999) (i.e., a calibration method based on a sigmoid function), isotonic regression (Zadrozny and Elkan, 2002), or more recently spline-based calibration (Lucena, 2018). One of the biggest challenges here is to find an appropriate mapping of the real-valued scores into the interval $[0, 1]$ so that the resulting probabilities can be interpreted as such – that is, in a way where they can be considered as well-calibrated posterior probabilities. In general, there is no guarantee for probabilistic classifiers that their resulting predictions are by default well-calibrated probabilities. For example, Niculescu-Mizil and Caruana (2005) show that some probabilistic classifiers tend to shift the probabilities away from 0 and 1, while others tend to shift them toward 0 and 1. In such situations, it is reasonable to correct this bias by recalibrating the predicted probabilities using the same strategies (e.g., Platt scaling or isotonic regression).

### 2.2.3. Remarks on Multiclass and Multilabel Classification

As for multiclass classification, it is also possible to construct a $m \times m$ confusion matrix for $m$ different classes. Similar to binary classification, several performance measures can be derived from such a $m \times m$ confusion matrix, with most of them in a one-versus-rest or one-versus-one fashion (cf. Flach, 2012, Ch. 3.1). Thus, many performance measures of binary classifiers can be extended to handle multiple classes, e.g., different variations of the multiclass AUC (Hand and Till, 2001; Ferri et al., 2009).

In the context of multilabel classification, there are observation-based and label-based performance measures. For the sake of completeness, the main differences and similarities between both types of measures are briefly explained here. As mentioned, in multilabel classification, each observation can belong to multiple labels at the same time. Therefore, the confusion matrix is computed for each observation separately, yielding the TP, FP, TN, and FN of individual observations. Based on such an observation-based confusion matrix, it is possible to calculate a performance measure for each observation separately, such as the $ACC$, $TPR$, or $FPR$. After that, the resulting values are averaged across all observations to obtain the final performance value.

An alternative for assessing the performance in multilabel classification are label-based performance measures. For this purpose, a confusion matrix is computed separately for each label. These label-based confusion matrices can be aggregated in two different ways to obtain label-based performance measures. Usually, this is achieved either by micro-averaging or by macro-averaging. Micro-averaging first computes an *averaged confusion matrix* by averaging each element (i.e., TP, FP, TN, and FN) across all label-based confusion matrices. After this, the desired performance measure is calculated only once, as the calculation is only based on this averaged confusion matrix. Conversely, the macro-averaging approach computes the desired performance measure for each label-based confusion matrix separately. Finally, it averages these label-based performance measures to obtain a macro-averaged performance value.

For both observation-based and label-based performance measures, there are also measures that are appropriate for discrete classifiers and measures that make more sense for probabilistic (or scoring) classifiers (cf. Zhang and Zhou, 2014; Charte and Charte, 2015). The contributing article in Chapter 3 compares several algorithm transformation methods with discrete classifiers using observation-based performance measures.

## 2.3. Performance Estimation

Obtaining reliable performance estimates based on available data is crucial for various applications. For example, in the context of this thesis, it is essential to assess the performance of prediction models (see Chapter 6) and the importance of features (see Chapter 7), to compare the performance of different algorithms (see benchmark studies in Chapter 3 and Chapter 4), and to support the field of meta-learning by sharing and retrieving reliable benchmark results (see Chapter 4 and Chapter 5).

Practitioners should always be aware of the overfitting issue when estimating the performance of a machine learning algorithm. As illustrated in Figure 2.1 and mentioned in Section 2.1.1, the overfitting issue suggests that several flexible algorithms are able to learn the functional relationship $f$ on the provided training data almost perfectly. However, the resulting prediction model $\hat{f}$ does not necessarily generalize well on new unobserved data. For this reason, assessing the performance of a prediction model based on the same data that was used to fit the model (i.e., the training data) usually yields a misleading (i.e., over-optimistic) performance estimate of the fitted model (cf. Efron, 1983). To overcome this issue, all existing performance estimation approaches mimic the situation of evaluating the performance on new unobserved data. This is achieved by dividing the available data into training data and test data (repeatedly). Figure 2.4 illustrates a general scheme to estimate the performance in supervised machine learning. On a unified high-level view, this involves three steps: a data splitting or resampling procedure, followed by the learning process and lastly the evaluation process. Two different quantities can be estimated with this general procedure, depending on whether the data splitting (and consequently also the model fitting) is repeated or not. More precisely, there is a difference between estimating the performance of a *prediction model* (i.e., the *conditional generalization error*) and estimating the performance of a *machine learning algorithm* (i.e., the *expected generalization error*).

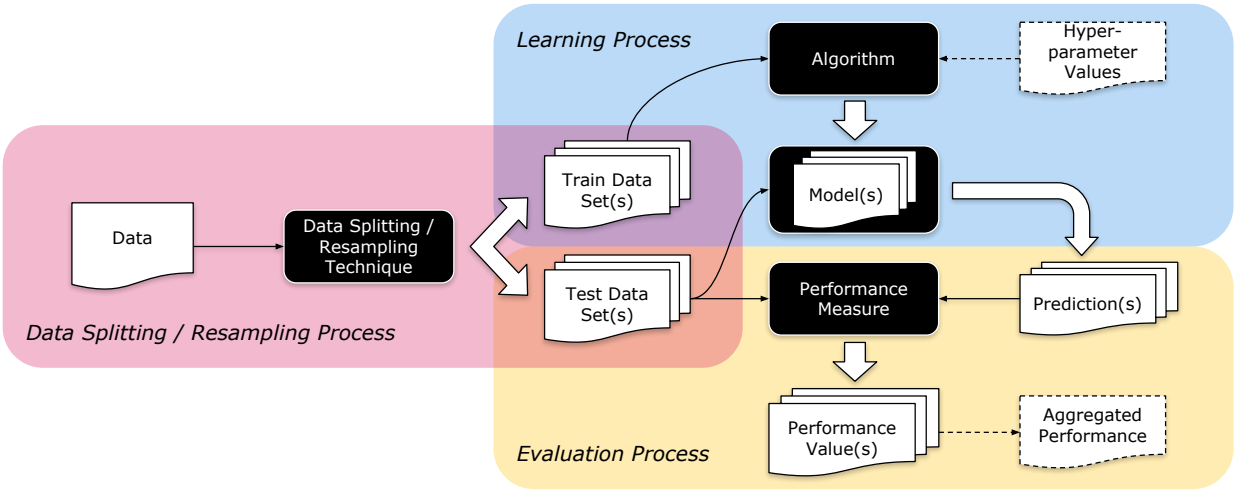



Figure 2.4.: A general scheme to estimate the performance: The data splitting / resampling process splits the available data into (a set of) training data and test data. The algorithm is trained on each training data and produces associated models (learning process). Each model along with its associated test data produces predictions. The performance measure calculates a performance value on each test data using the actual target values and their associated predictions. Multiple performance values are then aggregated to estimate the overall performance by a scalar value (evaluation process).

### 2.3.1. Conditional Generalization Error

Formally, the conditional generalization error of a fixed prediction model $\hat{f}_{\mathcal{D}}$ trained on some dataset $\mathcal{D}$ is defined as

$$GE(\hat{f}_{\mathcal{D}}, \mathcal{P}) = \mathbb{E}(L(\hat{f}_{\mathcal{D}}(X), Y)). \tag{2.7}$$

Here, $L$ is the loss function used to assess the performance of the model, which is implicitly defined through the choice of the performance measure as described in the previous section. Ideally, the performance is assessed on independently drawn observations from $\mathcal{P}$ that were not used to train the prediction model. Unfortunately, $\mathcal{P}$ is unknown, and it is not possible to directly draw any new observations from it. Thus, the performance is usually estimated by splitting the available data to mimic the presence of new unobserved data.

The holdout method is the most straightforward data-splitting approach to address this issue. Based on a pre-defined splitting ratio, the available data $\mathcal{D}$ is divided into two disjoint sets, namely the training data $\mathcal{D}_{\text{train}}$ and test data $\mathcal{D}_{\text{test}} = \mathcal{D} \setminus \mathcal{D}_{\text{train}}$. According to Figure 2.4, estimating the performance is then based on two further steps. First, during the learning process, the machine learning algorithm is trained on the training data $\mathcal{D}_{\text{train}}$. In the evaluation process, the resulting prediction model $\hat{f}_{\mathcal{D}_{\text{train}}}$ is then evaluated on the remaining test data, which contains observations not used during the learning process. Consequently, the holdout method estimates the conditional generalization error by

$$\widehat{GE}(\hat{f}_{\mathcal{D}_{\text{train}}}, \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} L(\hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}), y). \tag{2.8}$$

The holdout method preserves the test data for assessing the performance and does not use the available data $\mathcal{D}$ efficiently. Therefore, in order to obtain reliable performance estimates, the

holdout method usually requires the available data $\mathcal{D}$ to be large, as the choice of the splitting ratio involves a bias-variance trade-off (cf. Japkowicz and Shah, 2011, Ch. 5). This means that on the one hand, the holdout method requires enough training data to learn a representative model and to prevent the algorithm from learning less due to fewer observations, which would lead to a pessimistic bias. On the other hand, it is also crucial to have enough test data for the evaluation process to obtain a reliable estimate. Otherwise, the estimate introduces a huge variance. Furthermore, it is important to keep in mind that the holdout estimate is always conditional on the training data $\mathcal{D}_{\text{train}}$, since the algorithm used this training data to induce the prediction model $\hat{f}_{\mathcal{D}_{\text{train}}}$. Therefore, any conclusion that can be drawn from this estimate should refer to the prediction model and not to the algorithm itself. The contribution in Chapter 7 uses exactly this approach since the primary goal there was to infer conclusions about the importance of the features captured by an already fitted prediction model.

### 2.3.2. Expected Generalization Error

An algorithm $a$ usually induces different prediction models when applied to different datasets, even if the algorithm itself is of deterministic nature. Consequently, the application of an algorithm $a$ to a given dataset $\mathcal{D}$ results in $a(\mathcal{D}) = \hat{f}_{\mathcal{D}}$.[4] Suppose that it is possible to generate datasets $\mathcal{D}$ of equal size $n$ from $\mathcal{P}$. To take into account the variability introduced by randomly drawing different datasets $\mathcal{D}$ of equal size $n$ from $\mathcal{P}$, the *expected generalization error* of an algorithm $a$ is usually defined by

$$GE(a, \mathcal{P}, n) = \mathbb{E}_{|\mathcal{D}|=n}(GE(a(\mathcal{D}), \mathcal{P})). \tag{2.9}$$

Consequently, the expected generalization error refers to the expectation of the *conditional generalization error* with respect to all possible datasets $\mathcal{D}$ of size $n$ that were generated from the distribution $\mathcal{P}$.

Because $\mathcal{P}$ is unknown, resampling techniques have to reuse the available data $\mathcal{D}$ to obtain a stable estimate of the generalization error. For this purpose, the available data $\mathcal{D}$ is split into multiple (i.e., $B$) different training data of approximately equal size $n_{\text{train}}$ and corresponding test data, which are denoted by $\mathcal{D}^b_{\text{train}}$ and $\mathcal{D}^b_{\text{test}}$ with $b = 1, \ldots, B$, respectively. During the learning process, the algorithm induces multiple models, one for each training data (see also Figure 2.4), i.e., $a(\mathcal{D}^b_{\text{train}}) = \hat{f}_{\mathcal{D}^b_{\text{train}}}$ with $b = 1, \ldots, B$. Each model makes predictions on each test data, which are then assessed using Equation (2.7) to obtain individual performance values of each model. An estimate of the algorithm's performance is the average of these performance values, i.e.,

$$\widehat{GE}(a, \mathcal{D}, n_{\text{train}}) = \frac{1}{B} \sum_{b=1}^{B} \widehat{GE}(\hat{f}_{\mathcal{D}^b_{\text{train}}}, \mathcal{D}^b_{\text{test}}), \tag{2.10}$$

where $n_{\text{train}}$ refers to the size of each training data.

All existing resampling techniques mainly differ in how they produce the training and test datasets. Japkowicz and Shah (2011) make a distinction between two types, namely resampling techniques that use observations from the test data only once and resampling techniques that use observations

[4] In general, many algorithms are parametrized and can be configured with hyperparameters $\boldsymbol{\alpha}$, that is, a more general notation would be $a(\mathcal{D}, \boldsymbol{\alpha})$. However, as this thesis does not focus on hyperparameter optimization, the notation of $a(\mathcal{D})$ is used.

from the test data multiple times. For example, $k$-fold cross-validation (which produces $k$ non-overlapping train-test splits) belongs to the former type. The latter type includes the repeated $k$-fold cross-validation, random subsampling (which can be seen as a repeated holdout), as well as the bootstrap, which was introduced in Efron (1979). The properties of different resampling techniques have been widely studied and discussed in the literature (cf. Bengio and Grandvalet, 2004; Molinaro et al., 2005; Kim, 2009).

## 2.4. Benchmark Experiments

### 2.4.1. Motivation

In the past few decades, researchers have made considerable efforts to develop new machine learning algorithms or to extend well-established ones. Among other objectives, the intention has often been to overcome the limitations of existing algorithms and to obtain more accurate predictions. For a newly proposed algorithm to gain wide acceptance and generate interest, it is indispensable to provide empirical studies that illustrate the behavior of the novel algorithm (including its advantages and disadvantages) compared to other competing algorithms. In the field of statistics and machine learning, such empirical studies are called benchmark experiments and are usually performed on a collection of datasets. The purpose is to compare several algorithms in different data situations regarding their predictive performance, runtime, or any other measure of interest.

One of the main goals when comparing algorithms is to rank them according to a pre-defined performance measure of interest and to identify the best algorithm among all competing ones. At first glance, this goal seems contradictory to the *no free lunch* (NFL) theorem (Schaffer, 1994; Wolpert, 1996), which states that there is no such *master algorithm*[5] that strictly outperforms any other algorithm when averaging the algorithms' performances across all possible and conceivable datasets. However, the NFL theorem considers a universe where everything is uniformly distributed and equally possible, including the distribution from which all conceivable datasets are generated. As pointed out in Giraud-Carrier (2008) in their definition of the *weak assumption of machine learning*, "the process that presents us with learning problems induces a non-uniform probability distribution over the possible functions." This quote emphasizes that, in the world we live in, some datasets (among all conceivable datasets) are more likely to occur than others. In particular, this point implies that we are not necessarily interested in finding a master algorithm that outperforms any other algorithm in all conceivable datasets. Instead, we are interested in finding one that is only relevant for datasets resulting from real-world applications, or even only from specific domains or subdomains. For this reason, the space of all conceivable datasets of the universe is usually restricted to a well-defined subset (i.e., to the domain of interest). Here, a common underlying structure within this domain is assumed. This restriction can even be very general, such that the well-defined subset can still be an uncountable infinite set. In any case, it is widely accepted that the existence of a master algorithm in such a *closed world assumption* does not violate the NFL theorem in the context of machine learning (cf. Giraud-Carrier and Provost, 2005; Giraud-Carrier, 2008).

[5] The use of the term *master algorithm* in this context is based on the work of Domingos (2015).

Much work has already been done in developing frameworks for the statistical inferential analysis of benchmark results. Many of these frameworks enable researchers to draw general conclusions about the ranking of algorithms based on benchmark results. Starting from the framework proposed by Hothorn et al. (2005) for the statistical analysis of single dataset-based benchmark results, several generalizations that also allow for a domain-based analysis (i.e., an analysis based on multiple datasets from the same domain) of such benchmark results have been proposed. Examples of these generalizations include the frameworks proposed in Eugster et al. (2012) and Boulesteix et al. (2015). Furthermore, Eugster et al. (2014) proposed a general framework based on a paired comparison model, which makes it possible to analyze the influence of dataset characteristics on the algorithms' performance based on benchmark results. This general framework is based on the Bradley-Terry model for paired comparisons (cf. Bradley and Terry, 1952; Casalicchio et al., 2015).

Many contributions of the present dissertation focus on a different and rather technical but equally important aspect of benchmark experiments. Specifically, there are at least two somewhat technical requirements before such benchmark experiments can be conducted. First, a software tool is required that provides access to several machine learning algorithms to be compared. Ideally, the software tool also includes a convenient infrastructure to assess the performance of these algorithms. The mlr (Bischl et al., 2016) package for R can be used to meet this requirement. Second, there is a need for easily accessible and well-documented collections of datasets, on which the corresponding algorithms are applied to solve a common underlying machine learning task (e.g., regression or classification tasks). The contribution in Chapter 5 presents such a first collection of curated datasets at least for classification tasks, and it encourages other researchers to create and share their own benchmarking suites on OpenML (Vanschoren et al., 2013). Furthermore, the R package introduced in Chapter 4 makes this collection of datasets and all other datasets on OpenML easily accessible and queryable directly from R. The package also includes a convenient infrastructure in R for sharing and retrieving results of previous benchmark experiments.

### 2.4.2. Reproducibility and Reusability

In order to make reliable conclusions about the ranking of the algorithms and to also address the issue of the "variability across datasets" (Boulesteix et al., 2015), benchmark experiments should preferably be based on a large number of datasets. Fortunately, for this purpose, there are already many freely accessible datasets available from repositories, such as the UCI Machine Learning Repository (Dheeru and Karra Taniskidou, 2017), the PMLB benchmarking suite (Olson et al., 2017), and the UCR time series classification archive (Chen et al., 2015). However, the problem often is that many benchmark experiments published in scientific journals are rarely fully reproducible. One of the reasons for this is that the source code of the benchmark experiments is not always made available in many publications. Consequently, it is often impossible for other researchers to reproduce, build upon, extend, or modify previous benchmark experiments. This claim is in line with the findings in Hothorn and Leisch (2011) and Hofner et al. (2016) concerning the reproducibility of experiments in scientific publications in general. Convenient tools such as Sweave (Leisch, 2002) and knitr (Xie, 2014) make it already very easy for authors to embed the textual descriptions, the data, and the source code that produces the computational results into one single document. With such a single document, it is possible to generate a fully reproducible article if the computational experiments are easy and fast to compute, which is the case, e.g., with a simple explorative data analysis. However, to make benchmark experiments reproducible

in the same way and without caching intermediate results, it would require much more time and computational resources to generate a fully reproducible article (cf. Braun and Ong, 2014). Therefore, there may be a need for additional tools or platforms such as OpenML that help with organizing, structuring, and mining information of benchmark experiments in a convenient and machine-readable way.

Many details of benchmark experiments such as the properties of the datasets (e.g., the number of observations) and the performance values of the benchmark results themselves are often reported in publications as tables or figures, which are representations that are not easily machine-readable. Thus, even if the source code is provided, it is still required to re-run the benchmark experiments in order to have access to all details. Furthermore, the benchmark results found in different publications will most probably lack a unified format as there is no widely accepted and standardized way to publish such results. Consequently, it is often not feasible to reuse previous benchmark results from different publications for other purposes, such as to perform statistical (meta-)analyses based on these benchmark results. To some extent, this slows down machine learning research since researchers are now required to extract, join, and unify the benchmark results from different sources or re-run the experiments if they want to reuse these results, e.g., to perform their own statistical (meta-)analyses. Having access to meta-information of benchmark experiments could reveal a better understanding of these algorithms. For example, by analyzing the properties of datasets that cause certain algorithms to work better. Such meta-information can be easily stored in an online machine learning platform. If many researchers jointly use such a platform and feed it with more information, it may be possible for the machine learning community to identify better practices much faster than by reading traditional publications. Furthermore, when practitioners face a new machine learning problem, it may be helpful to analyze previous benchmark experiments on similar problems to find suitable algorithms. Similar tasks have already been carried out in the field of meta-learning. Here, one of the main objectives is to exploit meta-knowledge to better understand the relation between algorithms and machine learning tasks or datasets of a specific domain. This meta-knowledge can then be used to automatically propose suitable algorithms that are able to solve a new machine learning task, hence stimulating the *learning-to-learn* paradigm (Vilalta and Drissi, 2002).

The OpenML platform addresses many reproducibility and reusability issues of benchmark experiments. The platform allows researchers to share and organize their benchmark experiments online, and it facilitates reusing the benchmark results of other researchers. It also allows researchers to interact and collaborate with one another using their shared results. Furthermore, it is also possible to perform different meta-analyses based on previous benchmark results to obtain answers to various research questions such as whether certain characteristics of datasets are responsible for certain algorithms to perform better than other algorithms (cf. Bilalli et al., 2017).

## 2.5. Model-Agnostic Interpretability

### 2.5.1. Motivation

The hope in many machine learning applications is that a model may render help to understand something about the real world. However, it is important to remember that models are – if at all – only a simplification of certain parts of the real world. This is an issue that was already stressed by Box (1979) with the statement “all models are wrong, but some are useful.” Different

people may consider certain models more useful than others. Thus, the question arises as to what people understand to be a useful model. Does a useful model refer to a well-performing model that produces accurate predictions or does it refer to a simpler and therefore better interpretable model with probably less accurate predictions? In general, there is even a trade-off between model interpretability and model performance, which can be observed in many applications. If model interpretability is the primary concern, some practitioners tend to prefer simpler models. However, these simple models are often still improved through various means, such as preprocessing, feature engineering, and feature selection (Lipton, 2016). On the other hand, if model performance is of interest, some practitioners prefer to use machine learning models since they are known to produce more accurate predictions. Such machine learning models are often difficult to interpret and are consequently considered a black box (Caruana et al., 2015). Thus, a more appropriate question that can be asked is: How much is a user willing to pay for an interpretable model if the price to be paid is a drop in the performance of the model?

In general, not all applications require interpretable models. For example, in situations where unexpected results do not have serious consequences in the corresponding application, it may be reasonable to rely on well-performing black box models without worrying much about interpretability (Doshi-Velez and Kim, 2017). However, it can still make sense to optionally peek inside such a black box in order to extract valuable insights that have been captured by the model. Reasons for this include pure curiosity, the ability to make scientific explanations, or examination purposes, e.g., to check whether the model also captured effects that are expected in the real world (Miller, 2017). In other critical areas such as medicine or criminology, models are sometimes used to assist in making important decisions that can affect human life (Caruana et al., 2015; Krause et al., 2016). In such applications, it is essential to use an interpretable model that provides explanations for its decisions. Fortunately, there are many model-agnostic interpretation methods for such tasks that make it possible to explain the output of black box models in a similar way as intrinsically interpretable models. Examples of such interpretability methods include partial dependence (PD) plots (Friedman, 2001), functional ANOVA (Hooker, 2004, 2007), individual conditional expectation (ICE) plots (Goldstein et al., 2015), local interpretable model-agnostic explanations (LIME) (Ribeiro et al., 2016), SHapley Additive exPlanation (SHAP) values (Strumbelj and Kononenko, 2014; Lundberg and Lee, 2017), SHAP dependence plots (Lundberg et al., 2018), Sobol' indices (Sobol', 1990; Owen, 2014), the feature importance ranking measure (FIRM) (Zien et al., 2009), and the model reliance (Fisher et al., 2018). Many of these methods are also implemented in the R package iml (Molnar et al., 2018).

More recently, many researchers have been engaged with answering an even more fundamental question in the context of machine learning, namely the question of "what is interpretability?" (Lipton, 2016; Doshi-Velez and Kim, 2017; Miller, 2017; Biran and Cotton, 2017; Gilpin et al., 2018). For example, the definition in Gilpin et al. (2018) states that "for a system to be interpretable, it must produce descriptions that are simple enough for a person to understand using a vocabulary that is meaningful to the user." Clearly, descriptions or explanations that are understandable for one person may not necessarily be understandable for another person. Thus, looking for a single interpretability method that can be understood by anyone remains an open problem. The vast landscape of interpretability methods offers different types of explanations. For different applications, certain types of explanations are more appropriate to meet individual interpretation goals of practitioners. The next section presents an ontology that may help practitioners in choosing an appropriate interpretability method that meets their individual interpretation goals.

Furthermore, it is also shown how the methods introduced in Chapter 7 can be positioned within the proposed ontology.

### 2.5.2. An Ontology

An interpretability method ideally provides explanations that are appropriate for both the underlying application and the intended interpretation goal of the practitioner. Thus, the first step of the practitioner is to formulate the interpretation goal that he or she is most likely interested in. From a model-agnostic perspective, this is closely related to asking the question: Which *quantity of interest* that can be derived from a model deserves an explanation? Any quantity that can be derived from a model may be easier to understand for humans if it is possible to break it down into the individual contribution of each feature (or a set of features) to that quantity. The same objective has already been addressed by many interpretability methods, as these methods often decompose a certain quantity of interest into parts that are attributable to each feature (or set of features). In general, there are three frequently used quantities of interest in the literature that are often decomposed, namely 1) the model predictions (i.e., through feature effect methods), 2) the uncertainty or variability of the model predictions (i.e., through variance-based feature importance methods), or 3) the model performance (i.e., through performance-based feature importance methods). Specifically, many interpretability methods also often quantify or visualize how changes (or permutations) of one feature (or a set of features) affect the aforementioned quantities of interest. A similar categorization of interpretability methods has been described in Jiang and Owen (2002), Wei et al. (2015), Zhao and Hastie (2017), and Guidotti et al. (2018).

Figure 2.5 displays a general ontology. In its first dimension, it organizes common interpretability methods into the three *quantities of interest* mentioned above (i.e., feature effect methods, variance-based feature importance methods, and performance-based feature importance methods). The second dimension of the ontology refers to the *scope* of the interpretability method, that is, whether the method accounts for local or global explanations (Doshi-Velez and Kim, 2017). Local interpretability methods provide explanations for single observations (or a group of *similar* observations, e.g., neighboring observations that are close to each other). By contrast, global interpretability methods take into account all available observations or a (randomly selected) subset of these in order to provide explanations that are representative for the entire model. However, global explanations are often approximations of the overall behavior of the model since they are typically based on averaging local explanations. The third dimension in the ontology concerns the *output type* produced by the interpretability method. Here, the ontology further distinguishes between methods that provide summary statistics (e.g., quantitative values for each feature) and methods that also allow visualizations (e.g., a curve for each feature). Some interpretability methods also produce a surrogate model as intermediate output (e.g., the method LIME). However, since such surrogate models are ultimately used again to produce summary statistics or visualizations, this is not considered to be a third *output type* in the ontology.

It should be noted that the ontology described here should not be considered as exhaustive by any means. However, many interpretability methods can most probably be placed in the proposed ontology, or they can extend its design. For example, our contribution in Chapter 7 has extended the ontology for the performance-based feature importance in such a way that it allows practitioners to distinguish between a local and global performance-based feature importance similar to feature effects (this was not possible before our contribution). For illustration purposes, the

following paragraphs briefly describe how several model-agnostic interpretability methods can be placed into this ontology.

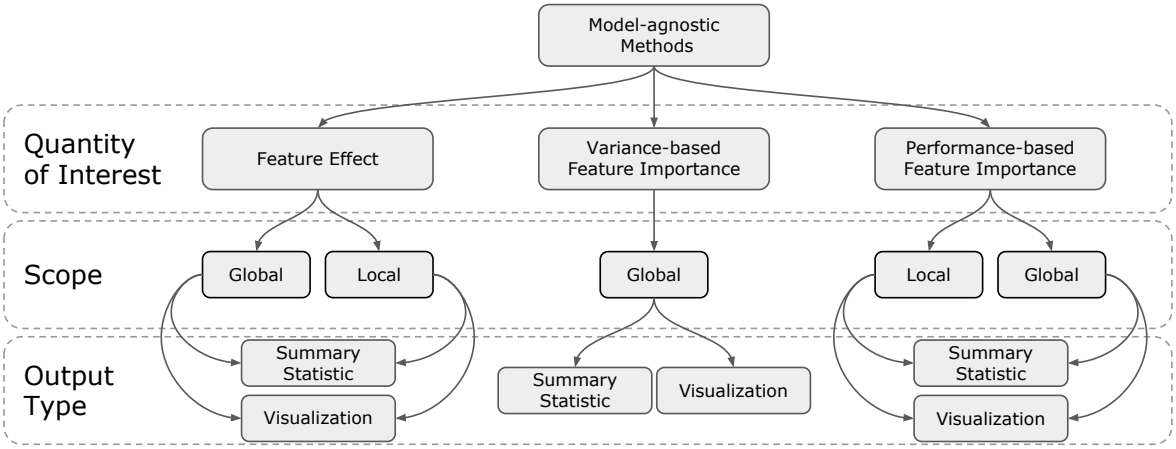



Figure 2.5.: A general ontology of model-agnostic interpretability methods.

### Feature Effects

In a simple linear regression model without interaction effects, the effect of a feature can be easily interpreted by examining the estimated regression coefficient of that feature. This is because the learned prediction function is based on the linear combination of all involved features of an observation $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)})^\top$, i.e.,

$$\hat{f}(\mathbf{x}^{(i)}) = \hat{\beta}_0 + x_1^{(i)}\hat{\beta}_1 + \cdots + x_p^{(i)}\hat{\beta}_p.$$

Thus, the coefficients $\hat{\beta}_j$ quantify the contribution of the $j$-th feature to the final prediction, and $\hat{\beta}_0$ is an intercept term. The coefficients are constant and do not vary across different observations. The interpretation of the feature effect is, therefore, very simple: if the value of the $j$-th feature increases by one unit (while holding all other feature values fixed), the expected value of the target outcome changes by the value of the regression coefficient $\hat{\beta}_j$. However, many machine learning algorithms are able to model more complex relationships such as high-order interactions or non-linear feature effects. Thus, the resulting prediction function is not directly human-interpretable. In such situations, the functional ANOVA (Hooker, 2004, 2007) approach can be used to decompose the high-dimensional prediction function into additive components that depend only on subsets of the feature space, i.e.,

$$\hat{f}(\mathbf{x}) = \underbrace{\hat{f}_\emptyset}_{\text{constant}} + \underbrace{\sum_{j=1}^{p} \hat{f}_j(\mathbf{x}_j)}_{\text{first order effects}} + \underbrace{\sum_{k \neq j}^{p} \hat{f}_{jk}(\mathbf{x}_j, \mathbf{x}_k)}_{\text{second order effects}} + \ldots = \sum_{k=0}^{p} \sum_{\substack{S \subseteq \{1,\ldots,p\}, \\ |S|=k}} \hat{f}_S(\mathbf{x}_S),$$

where $\mathbf{x}_j$ denotes the realizations of the $j$-th feature with $j = 1, \ldots, p$. Many model-agnostic methods often quantify or visualize the first order (i.e., main effects) or second order feature effects (i.e., interaction effects) on a local or global level and neglect the higher-order effects.

A local feature effect method is the ICE plot (Goldstein et al., 2015), which visualizes the marginal relationship of a feature and the target feature for individual observations by plotting a single curve for each observation. Typically, only the first order or second order effects are visualized while higher-order effects are neglected. SHAP values (Strumbelj and Kononenko, 2014; Lundberg and Lee, 2017) decompose the prediction of an individual observation into components attributable to each feature. This means that instead of neglecting the higher-order effects, SHAP values aim at fairly distributing the higher-order effects toward each individual feature based on the Shapley value from cooperative game theory (cf. Shapley, 1953). Another method with local scope is LIME (Ribeiro et al., 2016), which locally fits a surrogate model around a single observation based on artificially created observations in the neighborhood of the considered observation to be explained.

A global feature effect method is the PD plot (Friedman, 2001), as it visualizes the marginal relationship of a feature and the target feature across its range of feature values on a global level. Specifically, the PD plot is the pointwise average of the ICE curves. Examples of global methods that output a single quantitative summary statistic per feature include average marginal effects (AME) (Leeper, 2017) and other methods based on (averaging) partial derivatives (Ancona et al., 2018). However, such derivative-based methods are often restricted to differentiable algorithms, such as logistic regression or neural networks.

### Variance-Based Feature Importance

This paragraph briefly describes two types of variance-based feature importance methods. The first type is related to the idea of variance-based sensitivity analysis. Its aim is to decompose the overall variance of the model predictions into variance components attributable to each feature. For this task, it is possible to use methods from global sensitivity analysis (e.g., Sobol' indices (Sobol', 1990; Iooss and Lemaître, 2015)) or methods from cooperative game theory (e.g., Shapley values (Shapley, 1953; Owen, 2014)).

The second type is based on the idea that if varying the values of a feature does not significantly change model predictions (i.e., if the variance is low), the considered feature may be considered less important. Here, it is possible to use the variability of the PD plot of a feature as a measure of feature importance. As mentioned, the PD curve visualizes the feature effect on a global level. Thus, a "flat" curve indicates that changing the feature values does not significantly affect the feature effect. Therefore, such a feature can be considered less important. Two methods based on this idea are the feature importance ranking measure (FIRM) (Zien et al., 2009) and the method described in Greenwell et al. (2018).

### Performance-Based Feature Importance

The permutation importance introduced in Breiman (2001) belongs to the performance-based feature importance methods. However, the method is not a model-agnostic method, since it was initially introduced for random forests. Nevertheless, its main idea of permuting a feature and measuring the associated drop in performance can also be performed in a model-agnostic fashion. A model-agnostic counterpart of the permutation importance called model reliance was first described in Fisher et al. (2018). However, several implementations in various software tools have already been carried out before. For example, in Microsoft AzureML (Bleik, 2015; Team,

2016) and in the eli5 package for Python (Korobov and Lopuhin, 2017), there has already been an implementation for years.

The contribution in Chapter 7 introduces a performance-based feature importance measure with *local scope*. The idea is based on decomposing the model reliance of a feature into parts attributable to individual observations. Specifically, using the terminology regarding the *output type* from the proposed ontology, the individual conditional importance (ICI) curve is regarded as a visualization method, and the integral of the ICI curves is regarded as a summary statistic. Furthermore, the pointwise average of all ICI curves yields another novel visualization method with *global scope*, namely the partial importance (PI) curve, where its integral results in the model reliance. From this point of view, the model reliance can be categorized as a summary statistic with *global scope*. Chapter 7 also proposes another summary statistic with *global scope*, namely the Shapley feature importance measure (SFIMP), which aims at decomposing the model performance into parts attributable to each feature based on the Shapley value (Shapley, 1953). Thus, the proposed method is inspired by the two decomposition methods described above, namely the SHAP values (Strumbelj and Kononenko, 2014; Lundberg and Lee, 2017) for decomposing the feature effects and the variance-based feature importance method described in (Owen, 2014) for decomposing the variance of the model predictions. The only difference is that SFIMP decomposes the model performance instead of the variance of the model predictions or the model predictions themselves.

# Part I.

# Machine Learning Software in R

# 3. Multilabel Classification with R Package mlr

Chapter 3 describes several methods for multilabel classification that were also implemented into our R package mlr (Bischl et al., 2016). Furthermore, the methods are compared in a benchmark study on several multilabel datasets using different observation-based multilabel performance measures.

**Contributing article:**

Probst, P., Au, Q., Casalicchio, G., Stachl, C., and Bischl, B. (2017). Multilabel Classification with R Package mlr. *R Journal*, 9(1). *https://journal.r-project.org/archive/2017/RJ-2017-012/index.html*.

**Copyright information:**



**Author contributions:**

Philipp Probst and Quay Au prepared a first draft of the paper. Giuseppe Casalicchio reworked the paper and improved the mathematical notation. He also assisted in designing and running the benchmark study, as well as in analyzing and reporting the benchmark results. He further extended the benchmark study to accommodate the reviewers' comments. Clemens Stachl and Bernd Bischl added valuable input and suggested several notable modifications. All authors contributed to revisions of the paper.

**Supplementary material available at:**

- Supplementary material: `https://doi.org/10.6084/m9.figshare.3384802.v5`
- Tutorial: `https://mlr.mlr-org.com/articles/tutorial/multilabel.html`
- The mlr package:
  - Paper: `http://jmlr.org/papers/v17/15-066.html`
  - R package: `https://github.com/mlr-org/mlr`

# Multilabel Classification with R Package mlr

*by Philipp Probst, Quay Au, Giuseppe Casalicchio, Clemens Stachl and Bernd Bischl*

**Abstract** We implemented several multilabel classification algorithms in the machine learning package **mlr**. The implemented methods are binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking, which can be used with any base learner that is accessible in **mlr**. Moreover, there is access to the multilabel classification versions of **randomForestSRC** and **rFerns**. All these methods can be easily compared by different implemented multilabel performance measures and resampling methods in the standardized **mlr** framework. In a benchmark experiment with several multilabel datasets, the performance of the different methods is evaluated.

## Introduction

Multilabel classification is a classification problem where multiple target labels can be assigned to each observation instead of only one, like in multiclass classification. It can be regarded as a special case of multivariate classification or multi-target prediction problems, for which the scale of each response variable can be of any kind, for example nominal, ordinal or interval.

Originally, multilabel classification was used for text classification (McCallum, 1999; Schapire and Singer, 2000) and is now used in several applications in different research fields. For example, in image classification, a photo can belong to the classes *mountain* and *sunset* simultaneously. Zhang and Zhou (2008) and others (Boutell et al., 2004) used multilabel algorithms to classify scenes on images of natural environments. Furthermore, gene functional classifications is a popular application of multilabel learning in the field of biostatistics (Elisseeff and Weston, 2002; Zhang and Zhou, 2008). Additionally, multilabel classification is useful to categorize audio files. Music genres (Sanden and Zhang, 2011), instruments (Kursa and Wieczorkowska, 2014), bird sounds (Briggs et al., 2013) or even emotions evoked by a song (Trohidis et al., 2008) can be labeled with several categories. A song could, for example, be classified both as a *rock song* and a *ballad*.

An overview of multilabel classification was given by Tsoumakas and Katakis (2007). Two different approaches exist for multilabel classification. On the one hand, there are algorithm adaptation methods that try to adapt multiclass algorithms so they can be applied directly to the problem. On the other hand, there are problem transformation methods, which try to transform the multilabel classification into binary or multiclass classification problems.

Regarding multilabel classification software, there is the **mldr** (Charte and Charte, 2015) R package that contains some functions to get basic characteristics of specific multilabel datasets. The package is also useful for transforming multilabel datasets that are typically saved as ARFF-files (Attribute-Relation File Format) to data frames and vice versa. This is especially helpful because until now only the software packages MEKA (Read and Reutemann, 2012) and Mulan (Tsoumakas et al., 2011) were available for multilabel classification and both require multilabel datasets saved as ARFF-files to be executed. Additionally, the **mldr** package provides a function that applies the binary relevance or label powerset transformation method which transforms a multilabel dataset into several binary datasets (one for each label) or into a multiclass dataset using the set of labels for each observation as a single target label, respectively. However, there is no R package that provides a standardized interface for executing different multilabel classification algorithms. With the extension of the **mlr** package described in this paper, it will be possible to execute several multilabel classification algorithms in R with many different base learners.

In the following section of this paper, we will describe the implemented multilabel classification methods and then give a practical instruction of how to execute these algorithms in **mlr**. Finally, we present a benchmark experiment that compares the performance of all implemented methods on several datasets.

## Multilabel classification methods implemented in mlr

In this section, we present multilabel classification algorithms that are implemented in the **mlr** package (Bischl et al., 2016), which is a powerful and modularized toolbox for machine learning in R. The package offers a unified interface to more than a hundred learners from the areas classification, regression, cluster analysis and survival analysis. Furthermore, the package provides functions and tools that facilitate complex workflows such as hyperparameter tuning (see, e.g., Lang et al., 2015) and

feature selection that can now also be applied to the multilabel classification methods presented in this paper. In the following, we list the algorithm adaptation methods and problem transformation methods that are currently available in **mlr**.

### Algorithm adaptation methods

The **rFerns** (Kursa and Wieczorkowska, 2014) package contains an extension of the random ferns algorithm for multilabel classification. In the **randomForestSRC** (Ishwaran and Kogalur, 2016) package, multivariate classification and regression random forests can be created. In the classification case, the difference to standard random forests is that a composite normalized Gini index splitting rule is used. Multilabel classification can be achieved by using binary encoding for the labels.

### Problem transformation methods

Problem transformation methods try to transform the multilabel classification problem so that a simple binary classification algorithm, the so-called base learner, can be applied.

Let $n$ be the number of observations, let $p$ be the number of predictor variables and let $Z = \{z_1, \ldots, z_m\}$ be the set of all labels. Observations follow an unknown probability distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a $p-$dimensional input space of arbitrary measurement scales and $\mathcal{Y} = \{0,1\}^m$ is the target space. In our notation, $\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_p^{(i)}\right)^\top \in \mathcal{X}$ refers to the $i$-th observation and $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(n)}\right)^\top$ refers to the $j$-th predictor variable, for all $i = 1, \ldots, n$ and $j = 1, \ldots, p$. The observations $\mathbf{x}^{(i)}$ are associated with their multilabel outcomes $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \ldots, y_m^{(i)}\right)^\top \in \mathcal{Y}$, for all $i = 1, \ldots, n$. For all $k = 1, \ldots, m$, setting $y_k^{(i)} = 1$ indicates the relevance, i.e., the occurrence, of label $z_k$ for observation $\mathbf{x}^{(i)}$ and setting $y_k^{(i)} = 0$ indicates the irrelevance of label $z_k$ for observation $\mathbf{x}^{(i)}$. The set of all instances thus becomes $D = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \ldots, \left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right)\right\}$. Furthermore, $\mathbf{y}_k = \left(y_k^{(1)}, \ldots, y_k^{(n)}\right)^\top$ refers to the $k$-th target vector, for all $k = 1, \ldots, m$. Throughout this paper, we visualize multilabel classification problems in the form of tables ($n = 6$, $p = 3$, $m = 3$):

$D \mathrel{\hat{=}}$

| $\mathbf{x}_1$ | $\mathbf{x}_2$ | $\mathbf{x}_3$ | $\mathbf{y}_1$ | $\mathbf{y}_2$ | $\mathbf{y}_3$ |
|---|---|---|---|---|---|
| | | | 0 | 0 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 0 |
| | | | 1 | 1 | 1 |
| | | | 1 | 1 | 0 |
| | | | 1 | 1 | 0 |

(1)

The entries of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ can be of any (valid) kind, like continuous, binary, or categorical. The table in (1) visualizes this as an empty gray background. The target variables are indicated by a red background and can only take the binary values 0 or 1.

### Binary relevance

The binary relevance method (BR) is the simplest problem transformation method. BR learns a binary classifier for each label. Each classifier $C_1, \ldots, C_m$ is responsible for predicting the *relevance* of their corresponding label by a 0/1 prediction:

$$C_k : \mathcal{X} \longrightarrow \{0,1\}, \quad k = 1, \ldots, m$$

These binary prediction are then combined to a multilabel target. An unlabeled observation $\mathbf{x}^{(l)}$ is assigned the prediction $\left(C_1\left(\mathbf{x}^{(l)}\right), C_2\left(\mathbf{x}^{(l)}\right), \ldots, C_m\left(\mathbf{x}^{(l)}\right)\right)^\top$. Hence, labels are predicted independently of each other and label dependencies are not taken into account. BR has linear computational complexity with respect to the number of labels and can easily be parallelized.

### Modeling label dependence

In the problem transformation setting, the arguably simplest way (Montañés et al., 2014) to model label dependence is to condition classifier models not only on $\mathcal{X}$, but also on other label information. The idea is to augment the input space $\mathcal{X}$ with information of the output space $\mathcal{Y}$, which is available in the training step. There are different ways to realize this idea of augmenting the input space. In essence, they can be distinguished in the following way:

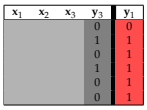
- Should the true label information be used? (True vs. predicted label information)
- For predicting one label $z_k$, should all other labels augment the input space, or only a subset of labels? (Full vs. partial conditioning)

### True vs. predicted label information

During the training of a classifier $C_k$ for the label $z_k$, the label information of other labels are available in the training data. Consequently, these true labels can directly be used as predictors to train the classifier. Alternatively, the predictions that are produced by some classifier can be used instead of the true labels.

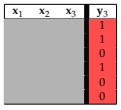
A classifier, which is trained on additional labels as predictors, needs those additional labels as input variables. Since these labels are not available at prediction time, they need to be predicted first. When the true label information is used to augment the feature space in the training of a classifier, the assumption that the training data and the test data should be identically distributed is violated (Senge et al., 2013). If the true label information is used in the training data and the predicted label information is used in the test data, the training data is not representative for the test data. However, experiments (Montañés et al., 2014; Senge et al., 2013) show that none of these methods should be dismissed immediately. Note that we use the superscript "true" or "pred" to emphasize that a classifier $C_k^{\text{true}}$ or $C_k^{\text{pred}}$ used true labels or predicted labels as additional predictors during training, respectively.

Suppose there are $n = 6$ observations with $p = 3$ predictors and $m = 3$ labels. The true label $\mathbf{y}_3$ shall be used to augment the feature space of a binary classifier $C_1^{\text{true}}$ for label $\mathbf{y}_1$. $C_1^{\text{true}}$ is thus trained on all predictors and the true label $\mathbf{y}_3$. The binary classification task for label $\mathbf{y}_1$ is therefore:

Train $C_1^{\text{true}}$ on  to predict $\mathbf{y}_1$ (2)



For an unlabeled observation $\mathbf{x}^{(l)}$, only the three predictor variables $x_1^{(l)}, \ldots, x_3^{(l)}$ are available at prediction time. However, the classifier $C_1^{\text{true}}$ needs a 4-dimensional observation $\left(\mathbf{x}^{(l)}, y_3^{(l)}\right)$ as input. The input $y_3^{(l)}$ therefore needs to be predicted first. A new *level-1* classifier $C_3^{\text{lvl1}}$, which is trained on the set $D' = \cup_{i=1}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$, will make those predictions for $y_3^{(l)}$. The training task is:

Train $C_3^{\text{lvl1}}$ on $D' \mathrel{\hat{=}}$  to predict $\mathbf{y}_3$ (3)



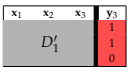
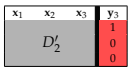
Therefore, for a new observation $\mathbf{x}^{(l)}$, the predicted label $\hat{y}_3^{(l)}$ is obtained by using $C_3^{\text{lvl1}}$ on $\mathbf{x}^{(l)}$. The final prediction for $y_1^{(l)}$ is then obtained by using $C_1^{\text{true}}$ on $\left(\mathbf{x}^{(l)}, \hat{y}_3^{(l)}\right)$.

The alternative to (2) would be to use predicted labels $\hat{\mathbf{y}}_3$ instead of true labels $\mathbf{y}_3$. These labels should be produced by means of an out-of-sample prediction procedure (Senge et al., 2013). This can be done by an internal leave-one-out cross-validation procedure, which can of course be computationally intensive. Because of this, coarser resampling strategies can be used. As an example, an internal 2-fold cross-validation will be shown here. Again, let $D' = \cup_{i=1}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$ be the set of all predictor variables with $\mathbf{y}_3$ as target variable. Using 2-fold cross-validation, the dataset $D'$ is split into two parts $D'_1 = \cup_{i=1}^{3} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$ and $D'_2 = \cup_{i=4}^{6} \left\{\left(\mathbf{x}^{(i)}, y_3^{(i)}\right)\right\}$:

 (4)



Two classifiers $C_{D'_1}$ and $C_{D'_2}$ are then trained on $D'_1$ and $D'_2$, respectively, for the prediction of $\mathbf{y}_3$:

Train $C_{D'_1}$ on  to predict $\mathbf{y}_3$, Train $C_{D'_2}$ on  to predict $\mathbf{y}_3$

Following the cross-validation paradigm, $D'_1$ is used as test set for the classifier $C_{D'_2}$, and $D'_2$ is used as a test set for $C_{D'_1}$:

$$C_{D'_2}: \begin{array}{|ccc|}\hline x_1 & x_2 & x_3 \\ \hline & D'_1 & \\ \hline \end{array} \mapsto \begin{array}{|c|}\hline \hat{y}_3 \\ \hline 1 \\ 0 \\ 0 \\ \hline \end{array}, \quad C_{D'_1}: \begin{array}{|ccc|}\hline x_1 & x_2 & x_3 \\ \hline & D'_2 & \\ \hline \end{array} \mapsto \begin{array}{|c|}\hline \hat{y}_3 \\ \hline 0 \\ 0 \\ 1 \\ \hline \end{array}$$

These predictions are merged for the final predicted label $\hat{\mathbf{y}}_3$, which is used to augment the feature space. The classifier $C_1^{\text{pred}}$ is then trained on that augmented feature space:

$$\text{Train } C_1^{\text{pred}} \text{ on } \begin{array}{|cccc|}\hline x_1 & x_2 & x_3 & \hat{y}_3 \\ \hline & & & 1 \\ & & & 0 \\ & & & 0 \\ & & & 0 \\ & & & 0 \\ & & & 1 \\ \hline \end{array} \begin{array}{|c|}\hline y_1 \\ \hline 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \hline \end{array} \text{ to predict } \mathbf{y}_1 \tag{5}$$

The prediction phase is completely analogous to (3). It is worthwhile to mention that the level-1 classifier $C_3^{\text{lvl1}}$, which will be used to obtain predictions $\hat{\mathbf{y}}_3$ at prediction time, is trained on the whole set $D' = D'_1 \cup D'_2$, following Simon (2007).

### Full vs. partial conditioning

Recall the set of all labels $Z = \{z_1, \ldots, z_m\}$. The prediction of a label $z_k$ can either be conditioned on all remaining labels $\{z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_m\}$ (*full conditioning*) or just on a subset of labels (*partial conditioning*). The only method for partial conditioning, which is examined in this paper, is the chaining method. Here, labels $z_k$ are conditioned on all previous labels $\{z_1, \ldots, z_{k-1}\}$ for all $k = 1, \ldots, m$. This sequential structure is motivated by the product rule of probability (Montañés et al., 2014):

$$P\left(\mathbf{y}^{(i)} \middle| \mathbf{x}^{(i)}\right) = \prod_{k=1}^{m} P\left(y_k^{(i)} \middle| \mathbf{x}^{(i)}, y_1^{(i)}, \ldots, y_{k-1}^{(i)}\right) \tag{6}$$

Methods that make use of this chaining structure are e.g., *classifier chains* or *nested stacking* (these methods will be discussed further below).

To sum up the discussions above: there are four ways in modeling label dependencies through conditioning labels $z_k$ on other labels $z_\ell$, $k \neq \ell$. They can be distinguished by the subset of labels, which are used for conditioning, and by the use of predicted or real labels in the training step. In Table 1 we show the four methods, which implement these ideas and describe them consequently.

| | True labels | Pred. labels |
|---|---|---|
| Partial cond. | Classifier chains | Nested stacking |
| Full cond. | Dependent binary relevance | Stacking |

**Table 1:** Distinctions in modeling label dependence and models

### Classifier chains

The classifier chains (CC) method implements the idea of using partial conditioning together with the true label information. It was first introduced by Read et al. (2011). CC selects an order on the set of labels $\{z_1, \ldots, z_m\}$, which can be formally written as a bijective function (permutation):

$$\tau : \{1, \ldots, m\} \longrightarrow \{1, \ldots, m\} \tag{7}$$

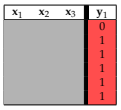
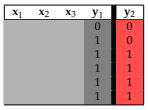
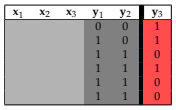
Labels will be chained along this order $\tau$:

$$z_{\tau(1)} \rightarrow z_{\tau(2)} \rightarrow \ldots \rightarrow z_{\tau(m)} \tag{8}$$

However, for this paper the permutation shall be $\tau = id$ (only for simplicity reasons). The labels therefore follow the order $z_1 \rightarrow z_2 \rightarrow \ldots \rightarrow z_m$. In a similar fashion to the binary relevance (BR) method, CC trains $m$ binary classifiers $C_k$, which are responsible for predicting their corresponding label $z_k$, $k = 1, \ldots, m$. The classifiers $C_k$ are of the form

$$C_k : \mathcal{X} \times \{0,1\}^{k-1} \longrightarrow \{0,1\}, \tag{9}$$

where $\{0,1\}^0 := \emptyset$. For a classifier $C_k$ the feature space is augmented by the true label information of all previous labels $z_1, z_2, \ldots, z_{k-1}$. Hence, the training data of $C_k$ consists of all observations $\left(\left(\mathbf{x}^{(i)}, y_1^{(i)}, y_2^{(i)}, \ldots, y_{k-1}^{(i)}\right), y_k^{(i)}\right)$, $i = 1, \ldots, n$, with the target $y_k^{(i)}$. In the example from above, this would look like:

Train $C_1$ on  Train $C_2$ on  Train $C_3$ on  (10)

At prediction time, when an unlabeled observation $\mathbf{x}^{(l)}$ is labeled, a prediction $\left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$ is obtained by successively predicting the labels along the chaining order:

$$\begin{aligned} \hat{y}_1^{(l)} &= C_1\left(\mathbf{x}^{(l)}\right) \\ \hat{y}_2^{(l)} &= C_2\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}\right) \\ &\vdots \\ \hat{y}_m^{(l)} &= C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_{m-1}^{(l)}\right) \end{aligned} \tag{11}$$

The authors of Senge et al. (2013) summarize several factors, which have an impact on the performance of CC:

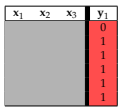
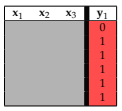
- *The length of the chain.* A high number $(k-1)$ of preceding classifiers in the chain comes with a high potential level of feature noise for the classifier $C_k$. One may assume that the probability of a mistake will increase with the level of feature noise in the input space. Then the probability of a mistake will be reinforced along the chain, due to the recursive structure of CC.
- *The order of the chain.* Some labels may be more difficult to predict than others. The order of a chain can therefore be important for the performance. It can be advantageous to put simple to predict labels in the beginning and harder to predict labels more towards the end of the chain. Some heuristics for finding an optimal chain ordering have been proposed in da Silva et al. (2014); Read et al. (2013). Alternatively Read et al. (2011) developed an ensemble of classifier chains, which builds many randomly ordered CC-classifiers and put them on a voting scheme for a prediction. However, these methods are not subject of this article.
- *The dependency among labels.* For an improvement of performance through chaining, there should be a dependence among labels, CC cannot gain in case of label independence. However, CC is also only likely to lose if the binary classifiers $C_k$ cannot ignore the added features $\mathbf{y}_1, \ldots, \mathbf{y}_{k-1}$.

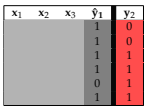
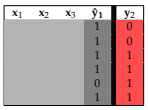
### Nested stacking

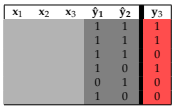
The nested stacking method (NST), first proposed in Senge et al. (2013), implements the idea of using partial conditioning together with predicted label information. NST mimicks the chaining structure of CC, but does not use real label information during training. Like in CC the chaining order shall be $\tau = id$ , again for simplicity reasons. CC uses real label information $\mathbf{y}_k$ during training and predicted labels $\hat{\mathbf{y}}_\mathbf{k}$ at prediction time. However, unless the binary classifiers are perfect, it is likely that $\mathbf{y}_k$ and $\hat{\mathbf{y}}_\mathbf{k}$ do not follow the same distribution. Hence, the key assumption of supervised learning, namely that the training data should be representative for the test data, is violated by CC. Nested stacking tries to overcome this issue by using predicted labels $\hat{\mathbf{y}}_\mathbf{k}$ instead of true labels $\mathbf{y}_k$.

NST trains $m$ binary classifiers $C_k$ on $D_k := \cup_{i=1}^n \left\{\left(\left(\mathbf{x}^{(i)}, \hat{y}_1^{(i)}, \ldots, \hat{y}_{k-1}^{(i)}\right), y_k^{(i)}\right)\right\}$, for all $k = 1, \ldots, m$. The predicted labels should be obtained by an internal out-of-sample method (Senge et al., 2013). How these predictions are obtained was already explained in the **True vs. Predicted Label Information** chapter. The prediction phase is completely analogous to (11).

The training procedure is visualized in the following with 2-fold cross-validation as an internal out-of-sample method:

Train $C_1$ on  Use 2-fold CV on  to obtain  (12)

Train $C_2$ on  Use 2-fold CV on  to obtain  (13)

Train $C_3$ on  (14)

The factors which impact the performance of CC (i.e., length and order of the chain, and the dependency among labels), also impact NST, since NST mimicks the chaining method of CC.

### Dependent binary relevance

The dependent binary relevance method (DBR) implements the idea of using full conditioning together with the true label information. DBR is built on two main hypotheses (Montañés et al., 2014):

(i) Taking conditional label dependencies into account is important for performing well in multilabel classification tasks.

(ii) Modeling and learning these label dependencies in an overcomplete way (take all other labels for modeling) may further improve model performance.

The first assumption is the main prerequisite for research in multilabel classification. It has been shown theoretically that simple binary relevance classifiers cannot achieve optimal performance for specific multilabel loss functions (Montañés et al., 2014). The second assumption, however, is harder to justify theoretically. Nonetheless, the practical usefulness of learning in an overcomplete way has been shown in many branches of (classical) single-label classification (e.g., ensemble methods (Dietterich, 2000)).

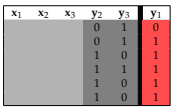
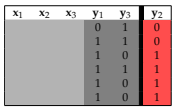
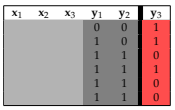
Formally, DBR trains $m$ binary classifiers $C_1, \ldots, C_m$ (as many classifiers as labels) on the corresponding training data

$$D_k = \cup_{i=1}^{n} \left\{ \left( \left( \mathbf{x}^{(i)}, y_1^{(i)}, \ldots, y_{k-1}^{(i)}, y_{k+1}^{(i)}, \ldots, y_m^{(i)} \right), y_k^{(i)} \right) \right\}, \tag{15}$$

$k = 1, \ldots, m$. Thus, each classifier $C_k$ is of the form

$$C_k : \mathcal{X} \times \{0,1\}^{m-1} \longrightarrow \{0,1\}.$$

Hence, for each classifier $C_k$ the true label information of all labels except $\mathbf{y}_k$ is used as augmented features. Again, here is a visualization with the example from above:

Train $C_1$ on  Train $C_2$ on  Train $C_3$ on  (16)

To make these classifiers applicable, when an unlabeled instance $\mathbf{x}^{(l)}$ needs to be labeled, the help of other multilabel classifiers is needed to produce predicted labels $\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}$ as additional features. The classifiers, which produce predicted labels as additional features, are called *base learners* (Montañés et al., 2014). Theoretically any multilabel classifier can be used as base learner. However, in this paper, the analysis is focused on BR as base learner only. The prediction of an unlabeled instance $\mathbf{x}^{(l)}$ formally works as follows:

(i) First level: Produce predicted labels by using the BR base learner:

$$C_{BR}\left(\mathbf{x}^{(l)}\right) = \left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$$

(ii) Second level, which is also called meta level (Montañés et al., 2014): Produce final prediction $\hat{\mathbf{y}}_k = \left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$ by applying DBR classifiers $C_1, \ldots, C_m$:

$$\begin{aligned} C_1\left(\mathbf{x}^{(l)}, \hat{y}_2^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{y}_1^{(l)} \\ C_2\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \hat{y}_3^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{y}_2^{(l)} \\ &\vdots \\ C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_{m-1}^{(l)}\right) &= \hat{y}_m^{(l)} \end{aligned}$$

### Stacking

Stacking (STA) implements the last variant of Table 1, namely the use of full conditioning together with predicted label information. Stacking is short for *stacked generalization* (Wolpert, 1992) and was first proposed in the multilabel context by Godbole and Sarawagi (2004). Like in classical stacking, for each label it takes predictions of several other learners that were trained in a first step to get a new learner to make predictions for the corresponding label. Both hypotheses on which DBR is built on also apply to STA, of course.

STA trains $m$ classifiers $C_1, \ldots, C_m$ on the corresponding training data

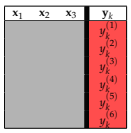
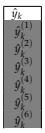
$$D_k = \cup_{i=1}^{n} \left\{ \left( \left( \mathbf{x}^{(i)}, \hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)} \right), y_k^{(i)} \right) \right\}, k = 1, \ldots, m. \tag{17}$$

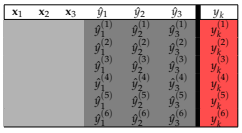
The classifiers $C_k$, $k = 1, \ldots, m$, are therefore of the following form:

$$C_k : \mathcal{X} \times \{0,1\}^m \longrightarrow \{0,1\}$$

Like in NST, the predicted labels should be obtained by an internal out-of-sample method (Sill et al., 2009). STA can be seen as the alternative to DBR using predicted labels (like NST is for CC). However, the classifiers $C_k, k = 1, \ldots, m$, are trained on all predicted labels $\hat{\mathbf{y}}_\mathbf{1}, \ldots, \hat{\mathbf{y}}_\mathbf{m}$ for the STA approach (in DBR the label $\mathbf{y}_k$ is left out of the augmented training set).

The training procedure is outlined in the following:

For i=1,2,3 use 2-fold CV on  to obtain  (18)

For i=1,2,3 train $C_k$ on  (19)

Like in DBR, STA depends on a BR base learner, to produce predicted labels as additional features. Again, the use of BR as a base learner is not mandatory, but it is the proposed method in Godbole and Sarawagi (2004).

The prediction of an unlabeled instance $\mathbf{x}^{(l)}$ works almost identically to the DBR case and is illustrated here:

(i) First level. Produce predicted labels by using the BR base learner:

$$C_{BR}\left(\mathbf{x}^{(l)}\right) = \left(\hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right)$$

(ii) Meta level. Apply STA classifiers $C_1, \ldots, C_m$:

$$\begin{aligned} C_1\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{y}_1^{(l)} \\ &\vdots \\ C_m\left(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_m^{(l)}\right) &= \hat{y}_m^{(l)} \end{aligned}$$

### Multilabel performance measures

Analogously to multiclass classification there exist multilabel classification performance measures. Six multilabel performance measures can be evaluated in **mlr**. These are: *Subset* 0/1 *loss*, *hamming loss*, *accuracy*, *precision*, *recall* and $F_1$-*index*. Multilabel performance measures are defined on a per instance basis. The performance on a test set is the average over all instances.

Let $D_{\text{test}} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right) \right\}$ be a test set with $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \ldots, y_m^{(i)}\right) \in \{0,1\}^m$ for all $i = 1, \ldots, n$. Performance measures quantify how good a classifier $C$ predicts the labels $z_1, \ldots, z_n$.

(i) The subset 0/1 loss is used to see if the predicted labels $C(\mathbf{x}^{(i)}) = \left(\hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)}\right)$ are equal to the actual labels $\left(y_1^{(i)}, \ldots, y_m^{(i)}\right)$:

$$\text{subset}_{0/1}\left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right) = \mathbb{1}_{\left(\mathbf{y}^{(i)} \neq C\left(\mathbf{x}^{(i)}\right)\right)} := \begin{cases} 1 & \text{if } \mathbf{y}^{(i)} \neq C\left(\mathbf{x}^{(i)}\right) \\ 0 & \text{if } \mathbf{y}^{(i)} = C\left(\mathbf{x}^{(i)}\right) \end{cases}$$

The subset 0/1 loss of a classifier $C$ on a test set $D_{\text{test}}$ thus becomes:

$$\text{subset}_{0/1}\left(C, D_{\text{test}}\right) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\mathbf{y}^{(i)} \neq C\left(\mathbf{x}^{(i)}\right)}$$

The subset 0/1 loss can be interpreted as the analogon of the mean misclassification error in multiclass classifications. In the multilabel case it is a rather drastic measure because it treats a mistake on a single label as a complete failure (Senge et al., 2013).

(ii) The hamming loss also takes into account observations where only some labels have been predicted correctly. It corresponds to the proportion of labels whose relevance is incorrectly predicted. For an instance $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right) = \left(\mathbf{x}^{(i)}, \left(y_1^{(i)}, \ldots, y_m^{(i)}\right)\right)$ and a classifier $C\left(\mathbf{x}^{(i)}\right) = \left(\hat{y}_1^{(i)}, \ldots, \hat{y}_m^{(i)}\right)$ this is defined as:

$$\text{HammingLoss}\left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right) = \frac{1}{m} \sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)}\right)}$$

If one label is predicted incorrectly, this accounts for an error of $\frac{1}{m}$. For a test set $D_{\text{test}}$ the hamming loss becomes:

$$\text{HammingLoss}(C, D_{\text{test}}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)}\right)}$$

The following measures are scores instead of loss function like the two previous ones.

(iii) The accuracy, also called Jaccard-Index, for a test set $D_{\text{test}}$ is defined as:

$$\text{accuracy}(C, D_{\text{test}}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ or } \hat{y}_k^{(i)}=1\right)}}$$

(iv) The precision for a test set $D_{\text{test}}$ is defined as:

$$\text{precision}(C, D_{\text{test}}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(\hat{y}_k^{(i)}=1\right)}}$$

(v) The recall for a test set $D_{\text{test}}$ is defined as:

$$\text{recall}(C, D_{\text{test}}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1\right)}}$$

(vi) For a test set $D_{\text{test}}$ the $F_1$-index is defined as follows:

$$F_1(C, D_{\text{test}}) = \frac{1}{n} \sum_{i=1}^{n} \frac{2 \sum_{k=1}^{m} \mathbb{1}_{\left(y_k^{(i)}=1 \text{ and } \hat{y}_k^{(i)}=1\right)}}{\sum_{k=1}^{m} \left(\mathbb{1}_{\left(y_k^{(i)}=1\right)} + \mathbb{1}_{\left(\hat{y}_k^{(i)}=1\right)}\right)}$$

The $F_1$-index is the harmonic mean of recall and precision on a per instance basis.

All these measures lie between 0 and 1. In the case of the subset 0/1 loss and the hamming loss the values should be low, in all other cases the scores should be high. Demonstrative definitions with sets instead of vectors can be seen in Charte and Charte (2015).

## Implementation

In this section, we briefly describe how to perform multilabel classifications in **mlr**. We provide small code examples for better illustration. A short tutorial is also available at http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html. The first step is to transform the multilabel dataset into a 'data.frame' in R. The columns must consist of vectors of features and one logical vector for each label that indicates if the label is present for the observation or not. To fit a multilabel classification algorithm in **mlr**, a multilabel task has to be created, where a vector of targets corresponding to the column names of the labels has to be specified. This task is an S3 object that contains the data, the target labels and further descriptive information. In the following example, the yeast data frame is extracted from the yeast.task, which is provided by the **mlr** package. Then the 14 label names of the targets are extracted and the multilabel task is created.

```
yeast = getTaskData(yeast.task)
labels = colnames(yeast)[1:14]
yeast.task = makeMultilabelTask(id = "multi", data = yeast, target = labels)
```

### Problem transformation methods

To generate a problem transformation method learner, a binary classification base learner has to be created with 'makeLearner'. A list of available learners for classifications in **mlr** can be seen at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/. Specific hyperparameter settings of the base learner can be set in this step through the 'par.vals' argument in 'makeLearner'. Afterwards, a learner for any problem transformation method can be created by applying the function 'makeMultilabel[...]Wrapper', where [...] has to be substituted by the desired problem transformation method. In the following example, two multilabel variants with rpart as base learner are created. The base learner is configured to output probabilities instead of discrete labels during prediction.

```
lrn = makeLearner("classif.rpart", predict.type = "prob")
multilabel.lrn1 = makeMultilabelBinaryRelevanceWrapper(lrn)
multilabel.lrn2 = makeMultilabelNestedStackingWrapper(lrn)
```

### Algorithm adaptation methods

Algorithm adaptation method learners can be created directly with 'makeLearner'. The names of the specific learner can be looked up at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/ in the multilabel section.

```
multilabel.lrn3 = makeLearner("multilabel.rFerns")
multilabel.lrn4 = makeLearner("multilabel.randomForestSRC")
```

### Train, predict and evaluate

Training and predicting on data can be done as usual in **mlr** with the functions 'train' and 'predict'. Learner and task have to be specified in 'train'; trained model and task or new data have to be specified in 'predict'.

```
mod = train(multilabel.lrn1, yeast.task, subset = 1:1500)
pred = predict(mod, task = yeast.task, subset = 1501:1600)
```

The performance of the prediction can be assessed via the function 'performance'. Measures are represented as S3 objects and multiple objects can be passed in as a list. The default measure for multilabel classification is the hamming loss (*multilabel.hamloss*). All available measures for multilabel classification can be shown by 'listMeasures' or looked up in the appendix of the tutorial page[1] (http://mlr-org.github.io/mlr-tutorial/release/html/measures/index.html).

[1] In the **mlr** package *precision* is named *positive predictive value* and *recall* is named *true positive rate*.

```
performance(pred, measures = list(multilabel.hamloss, timepredict))
multilabel.hamloss        timepredict
0.230              0.174
listMeasures("multilabel")
# [1] "multilabel.ppv" "timepredict"         "multilabel.hamloss" "multilabel.f1"
# [5] "featperc"       "multilabel.subset01" "timeboth"           "timetrain"
# [9] "multilabel.tpr" "multilabel.acc"
```

### Resampling

To properly evaluate the model, a resampling strategy, for example k-fold cross-validation, should be applied. This can be done in **mlr** by using the function 'resample'. First, a description of the subsequent resampling strategy, in this case three-fold cross-validation, is defined with 'makeResampleDesc'. The resample is executed by a call to the 'resample' function. The hamming loss is calculated for the binary relevance method.

```
rdesc = makeResampleDesc(method = "CV", stratify = FALSE, iters = 3)
r = resample(learner = multilabel.lrn1, task = yeast.task, resampling = rdesc,
measures = list(multilabel.hamloss), show.info = FALSE)
r
# Resample Result
# Task: multi
# Learner: multilabel.classif.rpart
# multilabel.hamloss.aggr: 0.23
# multilabel.hamloss.mean: 0.23
# multilabel.hamloss.sd: 0.00
# Runtime: 6.36688
```

### Binary performance

To calculate a binary performance measure like, e.g., the accuracy, the mean misclassification error (mmce) or the AUC for each individual label, the function 'getMultilabelBinaryPerformances' can be used. This function can be applied to a single multilabel test set prediction and also on a resampled multilabel prediction. To calculate the AUC, predicted probabilities are needed. These can be obtained by setting the argument 'predict.type = "prob"' in the 'makeLearner' function.

```
head(getMultilabelBinaryPerformances(r$pred, measures = list(acc, mmce, auc)))
#        acc.test.mean mmce.test.mean auc.test.mean
# label1     0.7389326      0.2610674     0.6801810
# label2     0.5908151      0.4091849     0.5935160
# label3     0.6512205      0.3487795     0.6631469
# label4     0.6921804      0.3078196     0.6965552
# label5     0.7517584      0.2482416     0.6748458
# label6     0.7343815      0.2656185     0.6054968
```

### Parallelization

In the case of a high number of labels and larger datasets, parallelization in the training and prediction process of the multilabel methods can reduce computation time. This can be achieved by using the package parallelMap in mlr (see also the tutorial section of parallelization: http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html). Currently, only the binary relevance method is parallelizable, the classifier for each label is trained in parallel, as they are independent of each other. The other problem transformation methods will also be parallelizable (as far as possible) soon.

```
library(parallelMap)
parallelStartSocket(2)
lrn = makeMultilabelBinaryRelevanceWrapper("classif.rpart")
mod = train(lrn, yeast.task)
pred = predict(mod, yeast.task)
```

## Benchmark experiment

In a similar fashion to Wang et al. (2014), we performed a benchmark experiment on several datasets in order to compare the performances of the different multilabel algorithms.

**Datasets**: In Table 2 we provide an overview of the used datasets. We retrieved most datasets from the Mulan Java library for multilabel learning[2] as well as from other benchmark experiments of multilabel classification methods. See Table 2 for article references. We uploaded all datasets to the open data platform OpenML (Casalicchio et al., 2017; Vanschoren et al., 2013), so they now can be downloaded directly from there. In some of the used datasets, sparse labels had to be removed in order to avoid problems during cross-validation. Several binary classification methods have difficulties when labels are sparse, i.e., a strongly imbalanced binary target class can lead to constant predictions for that target. That can sometimes lead to direct problems in the base learners (when training on constant class labels is simply not allowed) or, e.g., in classifier chains, when the base learner cannot handle constant features. Furthermore, one can reasonably argue that not much is to be learned for such a label. Hence, labels that appeared in less than 2% of the observations were removed. We computed *cardinality* scores (based on the remaining labels) indicating the mean number of labels assigned to each case in the respective dataset. The following description of the datasets refers to the final versions after removal of sparse labels.

- The first dataset (*birds*) consists of 645 audio recordings of 15 different vocalizing bird species (Briggs et al., 2013). Each sound can be assigned to various bird species.
- Another audio dataset (*emotions*) consists of 593 musical files with 6 clustered emotional labels (Trohidis et al., 2008) and 72 predictors. Each song can be labeled with one or more of the labels {*amazed-surprised*, *happy-pleased*, *relaxing-calm*, *quiet-still*, *sad-lonely*, *angry-fearful*}.
- The *genbase* dataset contains protein sequences that can be assigned to several classes of protein families (Diplaris et al., 2005). The entire dataset contains 1186 binary predictors.
- The *langLog*[3] dataset includes 998 textual predictors and was originally compiled in the doctorial thesis of Read (2010). It consists of 1460 text samples that can be assigned to one or more topics such as *language*, *politics*, *errors*, *humor* and *computational linguistics*.
- The UC Berkeley *enron*[4] dataset represents a subset of the original *enron*[5] dataset and consists of 1702 cases of emails with 24 labels and 1001 predictor variables (Klimt and Yang, 2004).
- A subset of the *reuters*[6] dataset includes 2000 observations for text classification (Zhang and Zhou, 2008).
- The *image*[7] benchmark dataset consists of 2000 natural scene images. Zhou and ling Zhang (2007) extracted 135 features for each image and made it publicly available as *processed* image dataset. Each observation can be associated with different label sets, where all possible labels are {*desert*, *mountains*, *sea*, *sunset*, *trees*}. About 22% of the images belong to more than one class. However, images belonging to three classes or more are very rare.
- The *scene* dataset is an image classification task where labels like *Beach, Mountain, Field, Urban* are assigned to each image (Boutell et al., 2004).
- The *yeast* dataset (Elisseeff and Weston, 2002) consists of micro-array expression data, as well as phylogenetic profiles of yeast, and includes 2417 genes and 103 predictors. In total, 14 different labels can be assigned to a gene, but only 13 labels were used due to label sparsity.
- Another dataset for text-classification is the *slashdot*[8] dataset (Read et al., 2011). It consists of article titles and partial blurbs. Blurbs can be assigned to several categories (e.g., *Science, News, Games*) based on word predictors.

**Algorithms**: We used all multilabel classification methods currently implemented in **mlr**: binary relevance (BR), classifier chains (CC), nested stacking (NST), dependent binary relevance (DBR) and stacking (STA) as well as algorithm adaption methods of the **rFerns** (RFERN) and **randomForestSRC** (RFSRC) packages. For DBR and STA the first level and meta level classifiers were equal. For CC and NST we chose random chain orders for each resample iteration.

[2] http://mulan.sourceforge.net/datasets-mlc.html
[3] http://languagelog.ldc.upenn.edu/nll/
[4] http://bailando.sims.berkeley.edu/enron_email.html
[5] http://www.cs.cmu.edu/~enron/
[6] http://lamda.nju.edu.cn/data_MIMLtext.ashx
[7] http://lamda.nju.edu.cn/data_MIMLimage.ashx
[8] http://slashdot.org

| Dataset | Reference | # Inst. | # Pred. | # Labels | Cardinality |
|---|---|---|---|---|---|
| birds* | Briggs et al. (2013) | 645 | 260 | 15 | 0.96 |
| emotions | Trohidis et al. (2008) | 593 | 72 | 6 | 1.87 |
| genbase* | Diplaris et al. (2005) | 662 | 112 | 16 | 1.20 |
| langLog* | Read (2010) | 1460 | 998 | 18 | 0.85 |
| enron* | Klimt and Yang (2004) | 1702 | 1001 | 24 | 3.12 |
| reuters | Zhang and Zhou (2008) | 2000 | 243 | 7 | 1.15 |
| image | Zhou and ling Zhang (2007) | 2000 | 135 | 5 | 1.24 |
| scene | Boutell et al. (2004) | 2407 | 294 | 6 | 1.07 |
| yeast* | Elisseeff and Weston (2002) | 2417 | 103 | 13 | 4.22 |
| slashdot* | Read et al. (2011) | 3782 | 1079 | 14 | 1.13 |

**Table 2:** Used benchmark datasets including number of instances, number of predictor, number of label and label cardinality. Datasets with an asterisk differ from the original dataset as sparse labels have been removed. The genbase dataset contained many constant factor variables, which were automatically removed by mlr.

**Base Learners**: We employed two different binary classification base learner for each problem transformation algorithm: random forest (rf) of the **randomForest** package (Liaw and Wiener, 2002) with ntree = 100 and adaboost (ad) from the **ada** package (Culp et al., 2012), each with standard hyperparameter settings.

**Performance Measures:** We used the six previously proposed performance measures. Furthermore, we calculated the reported values by means of a 10-fold cross-validation.

**Code:** For reproducibility, the complete code and results can be downloaded from Probst (2017). The R package **batchtools** (Bischl et al., 2015) was used for parallelization.

The results for hamming loss and $F_1$-index are illustrated in Figure 1. Tables 3 and 4 contain performance values with the best performing algorithms highlighted in blue. For all remaining measures one may refer to the Appendix. We did not perform any threshold tuning that would potentially improve some of the performance of the methods.

The results of the problem transformation methods in this benchmark experiment concur with the general conclusions and results in Montañés et al. (2014). The authors ran a similar benchmark study with penalized logistic regression as base learner. They concluded that, on average, DBR performs well in $F_1$ and accuracy. Also, CC outperform the other methods regarding the subset 0/1 loss most of the time. For the hamming loss measure they got mixed results, with no clear winner concordant to our benchmark results. As base learner, on average, adaboost performs better than random forest in our benchmark study.

Considering the measure $F_1$, the problem transformation methods DBR, CC, STA and NST outperform RFERN and RFSRC on most of the datasets and also almost always perform better than BR, which does not consider dependencies among the labels. RFSRC and RFERN only perform well on either precision or recall, but in order to be considered as good classifiers they should perform well on both. The generally poor performances of RFERN can be explained by the working mechanism of the algorithm which randomly chooses variables and split points at each split of a fern. Hence, it cannot deal with too many features that are useless for the prediction of the target labels.

## Summary

In this paper, we describe the implementation of multilabel classification algorithms in the R package **mlr**. The problem transformation methods binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking are implemented and can be used with any base learner that is accessible in **mlr**. Moreover, there is access to the multilabel classification versions of **randomForestSRC** and **RFerns**. We compare all of these methods in a benchmark experiment with several datasets and different implemented multilabel performance measures. The dependent binary relevance method performs well regarding the measures $F_1$ and *accuracy*. Classifier chains outperform the other methods in terms of the subset 0/1 loss most of the time. Parallelization is available for the binary relevance method and will be available soon for the other problem transformation methods. Algorithm adaptation methods and problem transformation methods that are currently not available can be incorporated in the current **mlr** framework easily. In our benchmark experiment we had to remove labels which occured too sparsely, because some algorithms crashed due to one class problems, which appeared during cross-validation. A solution to this problem and an implementation into the **mlr** framework is of great interest.

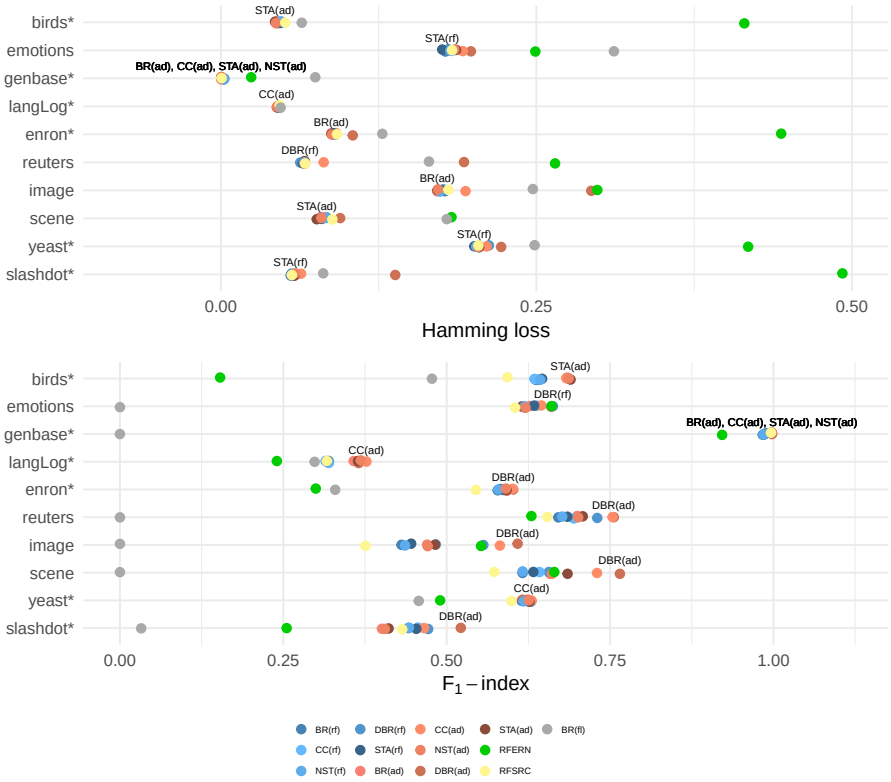



**Figure 1:** Results for hamming loss and $F_1$-index. The best performing algorithms are highlighted on the plot.

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.0477 | 0.0479 | 0.0475 | 0.0472 | 0.0468 | 0.0442 | 0.0441 | 0.0436 | 0.0431 | **0.0429** | 0.4148 | 0.0510 | 0.0641 |
| emotions | 0.1779 | 0.1832 | 0.1818 | 0.1801 | **0.1753** | 0.181 | 0.1916 | 0.1849 | 0.1981 | 0.1863 | 0.2492 | 0.1832 | 0.3114 |
| genbase* | 0.0021 | 0.0023 | 0.0025 | 0.0027 | 0.0023 | **0.0003** | **0.0003** | **0.0003** | 0.0004 | **0.0003** | 0.0240 | 0.0006 | 0.0748 |
| langLog* | 0.0464 | 0.0465 | 0.0467 | 0.0464 | 0.0466 | 0.0451 | **0.0442** | 0.0446 | 0.0447 | 0.0448 | 0.6673 | 0.0466 | 0.0473 |
| enron* | 0.0903 | 0.0904 | 0.0902 | 0.0909 | 0.0891 | **0.0874** | 0.0913 | 0.0881 | 0.1045 | 0.0877 | 0.4440 | 0.0919 | 0.1279 |
| reuters | 0.0663 | 0.0654 | 0.0661 | **0.0629** | 0.065 | 0.0666 | 0.0814 | 0.0664 | 0.1926 | 0.0664 | 0.2648 | 0.0668 | 0.1649 |
| image | 0.1774 | 0.1791 | 0.1737 | 0.1761 | 0.1754 | **0.1714** | 0.1939 | 0.1721 | 0.2935 | 0.1717 | 0.2983 | 0.1802 | 0.2472 |
| scene | 0.0836 | 0.0809 | 0.0832 | 0.0796 | 0.0799 | 0.0791 | 0.0821 | 0.0796 | 0.0945 | **0.076** | 0.1827 | 0.0884 | 0.1790 |
| yeast* | 0.2038 | 0.2044 | 0.2023 | 0.2123 | **0.2008** | 0.2048 | 0.2105 | 0.2038 | 0.2221 | 0.2046 | 0.4178 | 0.2040 | 0.2486 |
| slashdot* | 0.0558 | 0.0560 | 0.0559 | 0.0559 | **0.0554** | 0.059 | 0.0635 | 0.0586 | 0.1382 | 0.0582 | 0.4925 | 0.0562 | 0.0811 |

**Table 3:** Hamming loss

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.6369 | 0.6342 | 0.6433 | 0.64 | 0.6459 | 0.6835 | 0.683 | 0.6867 | 0.6846 | **0.6895** | 0.1533 | 0.5929 | 0.4774 |
| emotions | 0.6199 | 0.6380 | 0.6192 | **0.6625** | 0.6337 | 0.6274 | 0.6449 | 0.6206 | 0.6598 | 0.615 | 0.6603 | 0.6046 | 0.0000 |
| genbase* | 0.9885 | 0.9861 | 0.9855 | 0.9835 | 0.9861 | **0.9977** | **0.9977** | **0.9977** | 0.9962 | **0.9977** | 0.9214 | 0.9962 | 0.0000 |
| langLog* | 0.3192 | 0.3194 | 0.3148 | 0.3199 | 0.3167 | 0.3578 | **0.3772** | 0.3686 | 0.3653 | 0.3643 | 0.2401 | 0.3167 | 0.2979 |
| enron* | 0.5781 | 0.5822 | 0.5791 | 0.5866 | 0.5826 | 0.592 | 0.6009 | 0.5906 | **0.6017** | 0.5917 | 0.2996 | 0.5446 | 0.3293 |
| reuters | 0.6708 | 0.6944 | 0.6769 | 0.7303 | 0.6846 | 0.6997 | 0.7537 | 0.7012 | **0.7556** | 0.7082 | 0.6296 | 0.6541 | 0.0000 |
| image | 0.4308 | 0.4835 | 0.4362 | 0.5561 | 0.4456 | 0.47 | 0.5814 | 0.4709 | **0.6085** | 0.4824 | 0.5525 | 0.3757 | 0.0000 |
| scene | 0.6161 | 0.6420 | 0.6161 | 0.6563 | 0.6326 | 0.6585 | 0.73 | 0.661 | **0.765** | 0.685 | 0.6647 | 0.5729 | 0.0000 |
| yeast* | 0.6148 | 0.6294 | 0.6180 | 0.6195 | 0.6244 | 0.6238 | **0.63** | 0.6257 | 0.616 | 0.6266 | 0.4900 | 0.5991 | 0.4572 |
| slashdot* | 0.4415 | 0.4562 | 0.4422 | 0.4716 | 0.4535 | 0.4009 | 0.4654 | 0.4052 | **0.5216** | 0.411 | 0.2551 | 0.4320 | 0.0325 |

**Table 4:** $F_1$-index

## Bibliography


B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, and C. Weihs. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software*, 64(11): 1–25, 2015. URL https://doi.org/10.18637/jss.v064.i11. [p363]

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. [p352]

M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004. URL https://doi.org/10.1016/j.patcog.2004.03.009. [p352, 362, 363]

F. Briggs, H. Yonghong, R. Raich, and others. New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–8, 2013. URL https://doi.org/10.1109/mlsp.2013.6661934. [p352, 362, 363]

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the networked machine learning platform OpenML. *ArXiv e-prints*, 2017. [p362]

F. Charte and D. Charte. Working with multilabel datasets in R: The mldr package. *The R Journal*, 7(2): 149–162, 2015. [p352, 360]

M. Culp, K. Johnson, and G. Michailidis. *ada: An R Package for Stochastic Boosting*, 2012. URL https://cran.r-project.org/package=ada. [p363]

P. N. da Silva, E. C. Gonçalves, A. Plastino, and A. A. Freitas. Distinct chains for different instances: An effective strategy for multi-label classifier chains. In *European Conference, ECML PKDD 2014*, pages 453–468, 2014. URL https://doi.org/10.1007/978-3-662-44851-9_29. [p356]

T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000. URL https://doi.org/10.1007/3-540-45014-9_1. [p357]

S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas. Protein classification with multiple algorithms. In *Advances in Informatics*, pages 448–456. Springer-Verlag, 2005. URL https://doi.org/10.1007/11573036_42. [p362, 363]

A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 681–687. MIT Press, 2002. [p352, 362, 363]

S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data*, volume LNCS3056, pages 22–30, 2004. URL https://doi.org/10.1007/978-3-540-24775-3_5. [p358]

H. Ishwaran and U. B. Kogalur. *Random Forests for Survival, Regression and Classification (RF-SRC)*, 2016. URL http://cran.r-project.org/package=randomForestSRC. [p353]

B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. *Machine Learning: ECML 2004*, pages 217–226, 2004. URL https://doi.org/10.1007/978-3-540-30115-8_22. [p362, 363]

M. B. Kursa and A. A. Wieczorkowska. Multi-label ferns for efficient recognition of musical instruments in recordings. In *International Symposium on Methodologies for Intelligent Systems*, pages 214–223. Springer, 2014. URL https://doi.org/10.1007/978-3-319-08326-1_22. [p352, 353]

M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85 (1):62–76, 2015. URL https://doi.org/10.1080/00949655.2014.929131. [p352]

A. Liaw and M. Wiener. Classification and regression by randomForest. *R News: The Newsletter of the R Project*, 2(3):18–22, 2002. [p363]

A. McCallum. Multi-label text classification with a mixture model trained by EM. *AAAI'99 Workshop on Text Learning*, pages 1–7, 1999. [p352]

E. Montañés, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent binary relevance models for multi-label classification. *Pattern Recognition*, 47(3):1494–1508, 2014. URL https://doi.org/10.1016/j.patcog.2013.09.029. [p353, 354, 355, 357, 358, 363]

P. Probst. Multilabel classification with R package mlr. figshare. Code may be downloaded here, 2017. URL https://doi.org/10.6084/m9.figshare.3384802.v5. [p363]

J. Read. Scalable multi-label classification. *Hamilton, New Zealand: University of Waikato*, 2010. [p362, 363]

J. Read and P. Reutemann. Meka: A multi-label extension to WEKA, 2012. URL http://meka.sourceforge.net/. [p352]

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85:333–359, 2011. URL https://doi.org/10.1007/s10994-011-5256-5. [p355, 356, 362, 363]

J. Read, L. Martino, and D. Luengo. Efficient Monte Carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, (Mdc):1–36, 2013. URL https://doi.org/10.1016/j.patcog.2013.10.006. [p356]

C. Sanden and J. Z. Zhang. Enhancing multi-label music genre classification through ensemble techniques. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 705–714, 2011. URL https://doi.org/10.1145/2009916.2010011. [p352]

R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000. [p352]

R. Senge, J. J. del Coz Velasco, and E. Hüllermeier. Rectifying classifier chains for multi-label classification. *Space, 2 (8)*, 2013. [p354, 356, 359]

J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-Weighted Linear Stacking. *ArXiv e-prints*, 2009. [p358]

R. Simon. Resampling strategies for model assessment and selection. In W. Dubitzky, M. Granzow, and D. Berrar, editors, *Fundamentals of Data Mining in Genomics and Proteomics SE - 8*, pages 173–186. Springer-Verlag, 2007. URL https://doi.org/10.1007/978-0-387-47509-7_8. [p355]

K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. Multi-label classification of music into emotions. *ISMIR*, 8:325–330, 2008. URL https://doi.org/10.1186/1687-4722-2011-426793. [p352, 362, 363]

G. Tsoumakas and I. Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. URL https://doi.org/10.4018/jdwm.2007070101. [p352]

G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. P. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p352]

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. URL https://doi.org/10.1145/2641190.2641198. [p362]

H. Wang, X. Liu, B. Lv, F. Yang, and Y. Hong. Reliable multi-label learning via conformal predictor and random forest for syndrome differentiation of chronic fatigue in traditional chinese medicine. *PLoS ONE*, 9(6), 2014. URL https://doi.org/10.1371/journal.pone.0099565. [p362]

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. URL https://doi.org/10.1016/s0893-6080(05)80023-1. [p358]

M. L. Zhang and Z. H. Zhou. M3MIML: A maximum margin method for multi-instance multi-label learning. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 688–697, 2008. URL https://doi.org/10.1109/icdm.2008.27. [p352, 362, 363]

Z.-H. Zhou and M. ling Zhang. Multi-instance multilabel learning with application to scene classification. *Neural Information Processing Systems*, 40(7):2038–2048, 2007. [p362, 363]

*Philipp Probst*
*Department of Medical Informatics, Biometry and Epidemiology*
*LMU Munich*
*81377 Munich*

*Germany*
probst@ibe.med.uni-muenchen.de

*Quay Au*
*Department of Statistics*
*LMU Munich*
*80539 Munich*
*Germany*
quay.au@stat.uni-muenchen.de

*Giuseppe Casalicchio*
*Department of Statistics*
*LMU Munich*
*80539 Munich*
*Germany*
giuseppe.casalicchio@stat.uni-muenchen.de

*Clemens Stachl*
*Department of Psychology*
*LMU Munich*
*80802 Munich*
*Germany*
clemens.stachl@psy.lmu.de

*Bernd Bischl*
*Department of Statistics*
*LMU Munich*
*80539 Munich*
*Germany*
bernd.bischl@stat.uni-muenchen.de

## Appendices

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.4481 | 0.4481 | 0.4466 | 0.4497 | 0.4451 | **0.4156** | 0.4218 | 0.4171 | 0.4233 | 0.4202 | 0.9830 | 0.4777 | 0.5226 |
| emotions | 0.6846 | 0.6575 | 0.6728 | **0.6457** | 0.6626 | 0.6777 | 0.6643 | 0.7031 | 0.6845 | 0.6828 | 0.7992 | 0.6829 | 1.0000 |
| genbase* | 0.0333 | 0.0363 | 0.0393 | 0.0423 | 0.0363 | **0.0045** | **0.0045** | **0.0045** | 0.0060 | **0.0045** | 0.2115 | 0.0091 | 1.0000 |
| langLog* | 0.6836 | 0.6829 | 0.6884 | 0.6842 | 0.6856 | 0.6521 | **0.6349** | 0.6418 | 0.6438 | 0.6466 | 0.8589 | 0.6856 | 0.7021 |
| enron* | 0.8531 | 0.8413 | 0.8560 | 0.8408 | 0.8484 | 0.8496 | **0.819** | 0.8484 | 0.8320 | 0.8408 | 1.0000 | 0.8619 | 0.9982 |
| reuters | 0.3620 | 0.3405 | 0.3575 | 0.311 | 0.3515 | 0.349 | **0.2945** | 0.338 | 0.3495 | 0.3385 | 0.5830 | 0.3695 | 1.0000 |
| image | 0.6635 | 0.6150 | 0.6505 | 0.575 | 0.6445 | 0.63 | **0.539** | 0.6275 | 0.6225 | 0.619 | 0.8365 | 0.6955 | 1.0000 |
| scene | 0.4225 | 0.3926 | 0.4217 | 0.3835 | 0.4046 | 0.3913 | **0.3095** | 0.3805 | 0.3610 | 0.3648 | 0.7540 | 0.4570 | 1.0000 |
| yeast* | 0.8316 | 0.7600 | 0.8201 | 0.8167 | 0.8155 | 0.8304 | **0.7563** | 0.8134 | 0.8217 | 0.806 | 0.9338 | 0.8337 | 0.9855 |
| slashdot* | 0.6140 | 0.5994 | 0.6116 | **0.5859** | 0.6052 | 0.6489 | 0.5923 | 0.6449 | 0.6658 | 0.6396 | 0.9966 | 0.6142 | 0.9675 |

**Table 5:** Subset 0/1 loss

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.6153 | 0.6126 | 0.6197 | 0.6169 | 0.6232 | 0.6589 | 0.657 | 0.6604 | 0.6581 | **0.6621** | 0.0999 | 0.5753 | 0.4774 |
| emotions | 0.5453 | 0.5649 | 0.5464 | **0.5849** | 0.5609 | 0.5519 | 0.5676 | 0.5408 | 0.5727 | 0.5427 | 0.5503 | 0.5332 | 0.0000 |
| genbase* | 0.9834 | 0.9806 | 0.9796 | 0.9773 | 0.9806 | **0.9972** | **0.9972** | **0.9972** | 0.9957 | **0.9972** | 0.8884 | 0.9950 | 0.0000 |
| langLog* | 0.3185 | 0.3188 | 0.3140 | 0.3188 | 0.3161 | 0.3553 | **0.3741** | 0.366 | 0.363 | 0.3615 | 0.1953 | 0.3161 | 0.2979 |
| enron* | 0.4693 | 0.4757 | 0.4694 | 0.4804 | 0.4742 | 0.483 | **0.4987** | 0.4824 | 0.4919 | 0.4847 | 0.1859 | 0.4394 | 0.2241 |
| reuters | 0.6625 | 0.6856 | 0.6682 | 0.7199 | 0.6754 | 0.6873 | **0.7414** | 0.6912 | 0.7197 | 0.6964 | 0.5620 | 0.6482 | 0.0000 |
| image | 0.4068 | 0.4585 | 0.4142 | 0.5225 | 0.4228 | 0.4446 | **0.5508** | 0.4458 | 0.5366 | 0.4564 | 0.4467 | 0.3578 | 0.0000 |
| scene | 0.6064 | 0.6333 | 0.6067 | 0.6463 | 0.6233 | 0.646 | 0.7201 | 0.6505 | **0.7313** | 0.6725 | 0.5513 | 0.5654 | 0.0000 |
| yeast* | 0.5091 | 0.5320 | 0.5138 | 0.514 | 0.5205 | 0.5182 | **0.5345** | 0.522 | 0.5068 | 0.5239 | 0.3674 | 0.4945 | 0.3361 |
| slashdot* | 0.4274 | 0.4421 | 0.4285 | 0.4569 | 0.4385 | 0.3883 | 0.4507 | 0.3925 | **0.4613** | 0.3982 | 0.1651 | 0.4202 | 0.0325 |

**Table 6:** Accuracy

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.2763 | 0.2752 | 0.2897 | 0.2859 | 0.2936 | 0.3755 | 0.3865 | 0.3772 | 0.3687 | 0.3784 | **0.8352** | 0.1949 | 0.0000 |
| emotions | 0.6197 | 0.6474 | 0.6187 | 0.6847 | 0.6358 | 0.6335 | 0.6708 | 0.6293 | 0.7189 | 0.6237 | **0.8276** | 0.6001 | 0.0000 |
| genbase* | 0.9846 | 0.9819 | 0.9809 | 0.9786 | 0.9819 | **0.9977** | **0.9977** | **0.9977** | 0.9962 | **0.9977** | 0.9962 | 0.9955 | 0.0000 |
| langLog* | 0.0334 | 0.0330 | 0.0270 | 0.0331 | 0.0308 | 0.0971 | 0.1191 | 0.1056 | 0.0995 | 0.102 | **0.9264** | 0.0301 | 0.0000 |
| enron* | 0.5426 | 0.5487 | 0.5421 | 0.5580 | 0.5466 | 0.5611 | 0.5902 | 0.5619 | 0.6314 | 0.5633 | **0.771** | 0.4959 | 0.2613 |
| reuters | 0.6733 | 0.6959 | 0.6801 | 0.7338 | 0.6875 | 0.7038 | 0.754 | 0.7046 | **0.9032** | 0.7123 | 0.8598 | 0.6559 | 0.0000 |
| image | 0.4192 | 0.4696 | 0.4228 | 0.5562 | 0.4335 | 0.4581 | 0.5691 | 0.4603 | **0.7787** | 0.4724 | 0.7374 | 0.3598 | 0.0000 |
| scene | 0.6148 | 0.6373 | 0.6134 | 0.6555 | 0.6306 | 0.6614 | 0.7243 | 0.6613 | 0.8174 | 0.6879 | **0.9173** | 0.5662 | 0.0000 |
| yeast* | 0.5722 | 0.6097 | 0.5788 | 0.6035 | 0.5874 | 0.5951 | 0.6229 | 0.5978 | 0.6104 | 0.6013 | **0.6296** | 0.5442 | 0.3365 |
| slashdot* | 0.4267 | 0.4412 | 0.4270 | 0.4574 | 0.4391 | 0.3834 | 0.4526 | 0.3868 | 0.6984 | 0.3931 | **0.8065** | 0.4094 | 0.0000 |

**Table 7:** Recall

| | BR(rf) | CC(rf) | NST(rf) | DBR(rf) | STA(rf) | BR(ad) | CC(ad) | NST(ad) | DBR(ad) | STA(ad) | RFERN | RFSRC | BR(fl) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| birds* | 0.8812 | 0.8889 | 0.8764 | **0.9056** | 0.8874 | 0.8461 | 0.8349 | 0.8401 | 0.8648 | 0.8605 | 0.0859 | 0.8996 | |
| emotions | 0.7627 | 0.7242 | 0.7499 | 0.7265 | **0.7644** | 0.7537 | 0.7014 | 0.739 | 0.6783 | 0.7347 | 0.5869 | 0.7577 | |
| genbase* | 0.9987 | 0.9987 | 0.9987 | 0.9987 | 0.9987 | **0.9995** | **0.9995** | **0.9995** | **0.9995** | **0.9995** | 0.8917 | **0.9995** | |
| langLog* | 0.7267 | **0.7356** | 0.7058 | 0.7207 | 0.6882 | 0.6874 | 0.7228 | 0.7133 | 0.7014 | 0.6965 | 0.0632 | 0.7233 | |
| enron* | 0.7283 | 0.7188 | 0.7305 | 0.7092 | 0.7331 | 0.7235 | 0.6807 | 0.7198 | 0.6371 | 0.7233 | 0.1973 | **0.7448** | 0.5135 |
| reuters | 0.9411 | 0.9168 | 0.9346 | 0.8995 | 0.9298 | 0.9014 | 0.7689 | 0.8983 | 0.7465 | 0.8931 | 0.5715 | **0.9562** | |
| image | 0.7899 | 0.7333 | 0.8029 | 0.7086 | 0.7865 | 0.7841 | 0.6281 | 0.7814 | 0.6036 | 0.7737 | 0.4813 | **0.83** | |
| scene | 0.9071 | 0.8956 | 0.9112 | 0.8917 | 0.9143 | 0.8936 | 0.81 | 0.8856 | 0.7879 | 0.8872 | 0.5662 | **0.9233** | |
| yeast* | 0.7372 | 0.7218 | 0.7351 | 0.7055 | 0.7389 | 0.7225 | 0.6947 | 0.7233 | 0.6827 | 0.7159 | 0.4361 | **0.7508** | 0.7478 |
| slashdot* | 0.8365 | 0.8127 | 0.8298 | 0.7927 | 0.8277 | 0.8119 | 0.6804 | 0.8161 | 0.5025 | 0.8196 | 0.1679 | **0.8366** | |

**Table 8:** Precision (For the featureless learner we have no precision results for several datasets. The reason is that the featureless learner does not predict any value in all observations in these datasets. Hence, the denominator in the precision formula is always zero. *mlr* predicts NA in this case.)

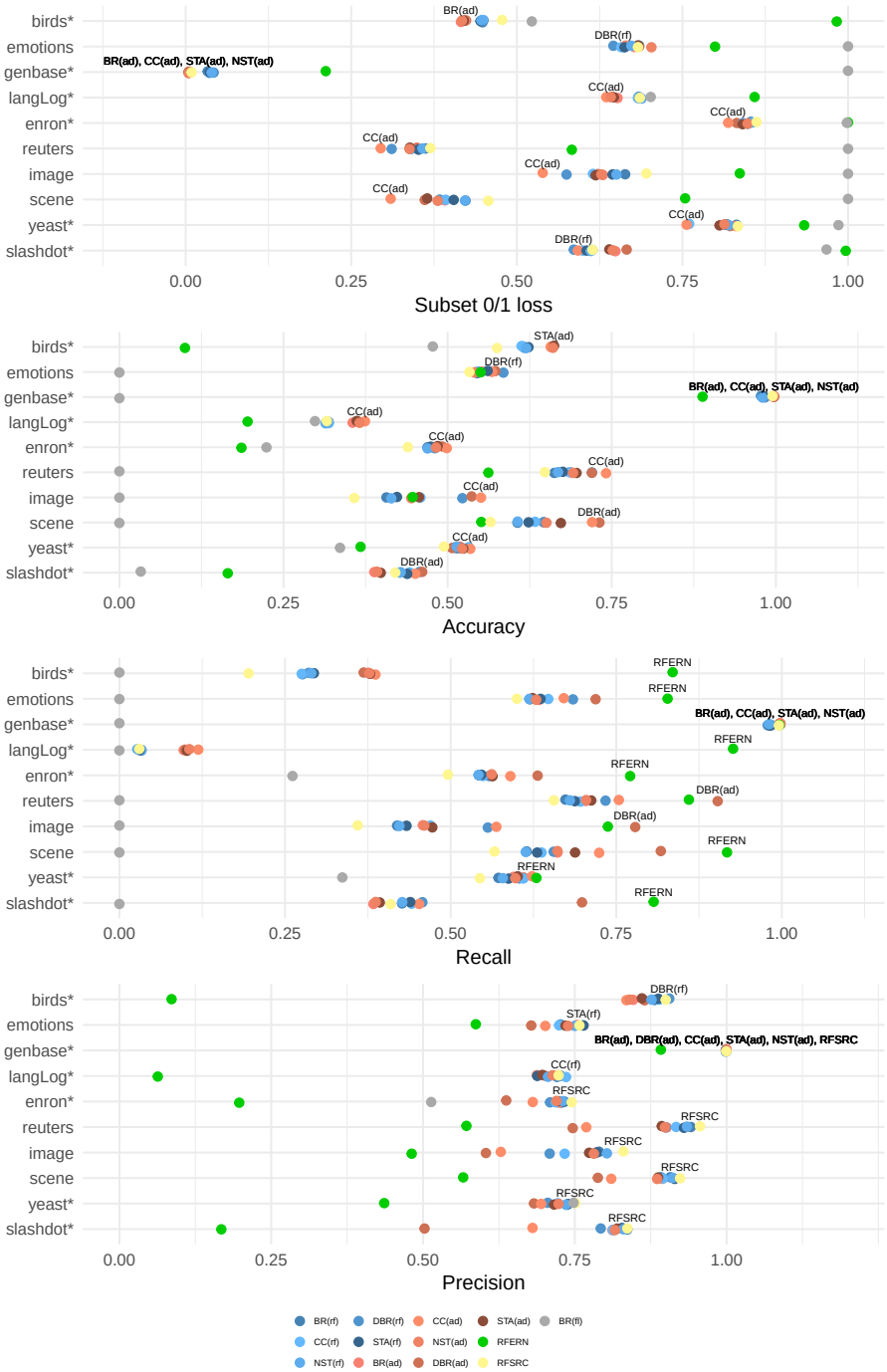



**Figure 2:** Results for the remaining measures.

# Part II.

# On Benchmark Experiments with OpenML

# 4. OpenML: An R Package to Connect to the Machine Learning Platform OpenML

Chapter 4 introduces the R package OpenML, which provides a simple interface for communicating with the OpenML server directly from within R and allows users to easily search, download, and upload datasets and benchmark results.

**Contributing article:**

Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., and Bischl, B. (2017). OpenML: An R Package to Connect to the Machine Learning Platform OpenML. *Computational Statistics*, pages 1–15. *https://doi.org/10.1007/s00180-017-0742-2*.



**Author contributions:**

Giuseppe Casalicchio prepared a first draft of the whole paper. Section 1 and 2 were mainly drafted by Joaquin Vanschoren. Bernd Bischl, Dominik Kirchhoff and Michel Lang designed and prepared a first draft of the package with some basic functionalities. Giuseppe Casalicchio took over the package development and reworked most parts of the package. He further continuously extended, tested and adapted the package due to the further ongoing development of the OpenML project. All authors added valuable input, proofread and contributed to revisions of the paper.

**Supplementary material available at:**

- R package: `https://github.com/openml/openml-r`
- Tutorial: `http://openml.github.io/openml-r`

Comput Stat
DOI 10.1007/s00180-017-0742-2

ORIGINAL PAPER

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

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

Received: 23 November 2016 / Accepted: 3 June 2017


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

**Keywords** Databases · Machine learning · R · Reproducible research

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

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

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

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

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

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

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

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

Published online: 19 June 2017

## 1 Introduction

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

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

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

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

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

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

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

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

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

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

## 2 The OpenML platform

The OpenML platform consists of several layers of software:

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

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

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

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

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

## 3 The mlr R package

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

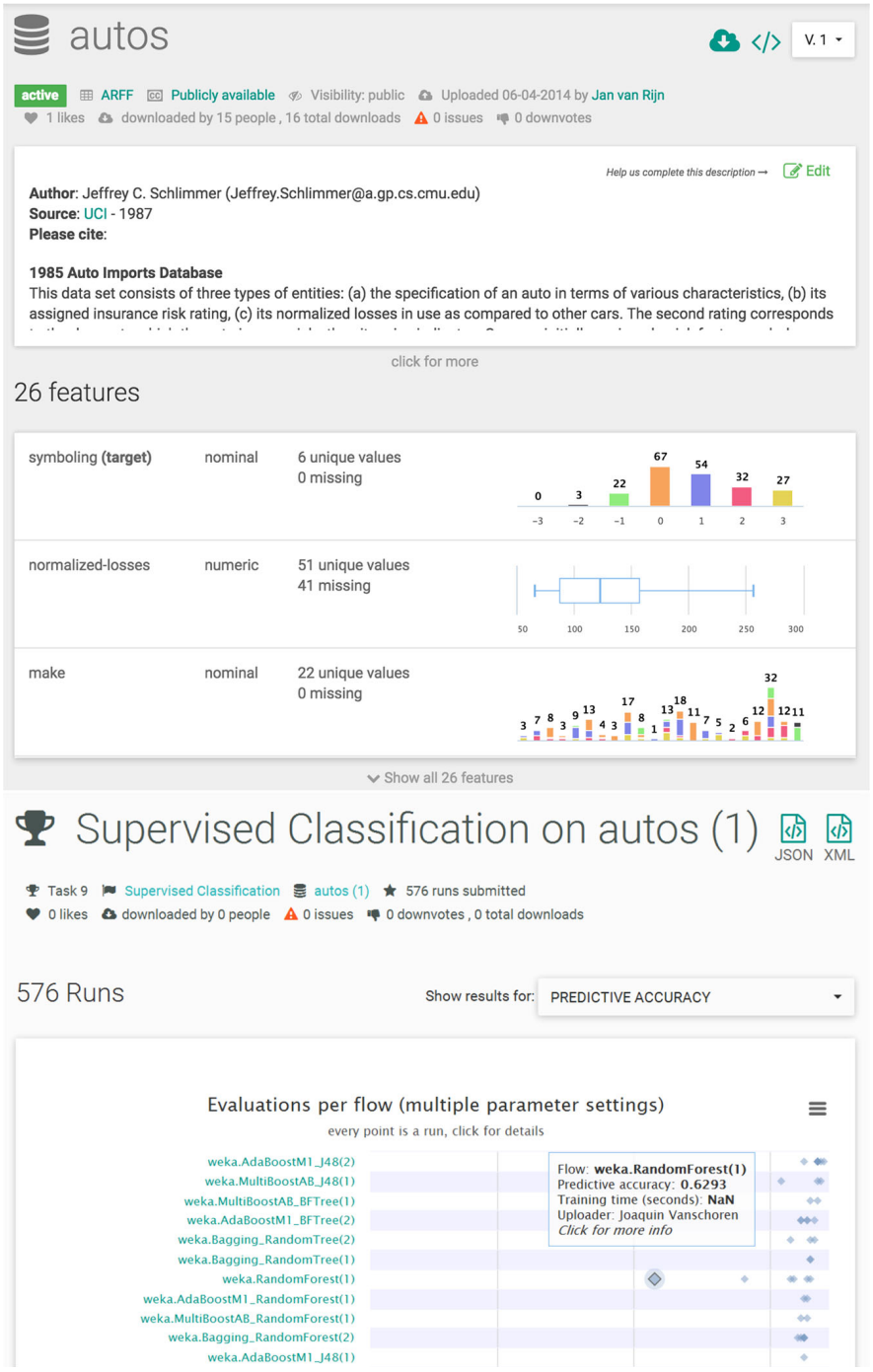



**Fig. 1** Screenshots of the OpenML website. The *top part* shows the data set 'autos', with wiki description and descriptive overview of the data features. The *bottom part* shows a classification task, with an overview of the best submitted flows with respect to the predictive accuracy as performance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

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

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

## 4 The OpenML R package

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

### 4.1 Installation and configuration

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

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

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

### 4.2 Listing information

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

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

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

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

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

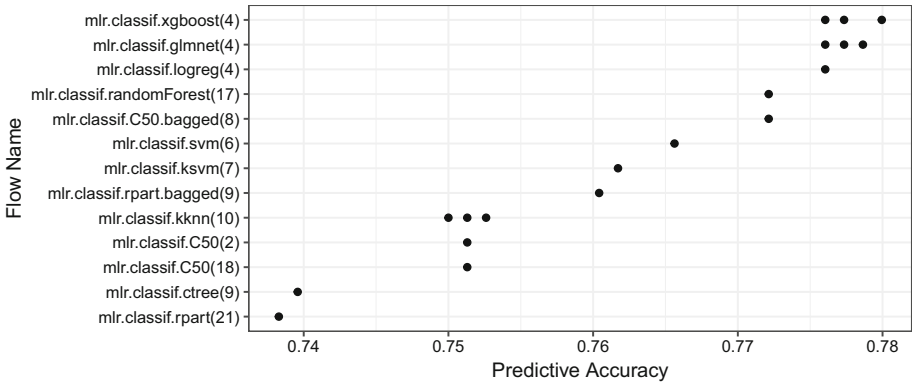



**Fig. 2** The predictive accuracy of some `mlr` flows on task 37. The numbers in *brackets* refer to the version of the flow. Multiple *dots* for the same flow refer to runs with different hyperparameter values for that flow

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

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

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

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

### 4.3 Downloading OpenML objects

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

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

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

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

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

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

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

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

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

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

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

### 4.4 Creating runs

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

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

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

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

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

### 4.5 Uploading and tagging

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

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

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

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

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

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

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

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

### 4.6 Further features

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

## 5 Case study

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

### 5.1 Creating learners

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

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

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

### 5.2 Querying OpenML

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

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

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

### 5.3 Evaluating results

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

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

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

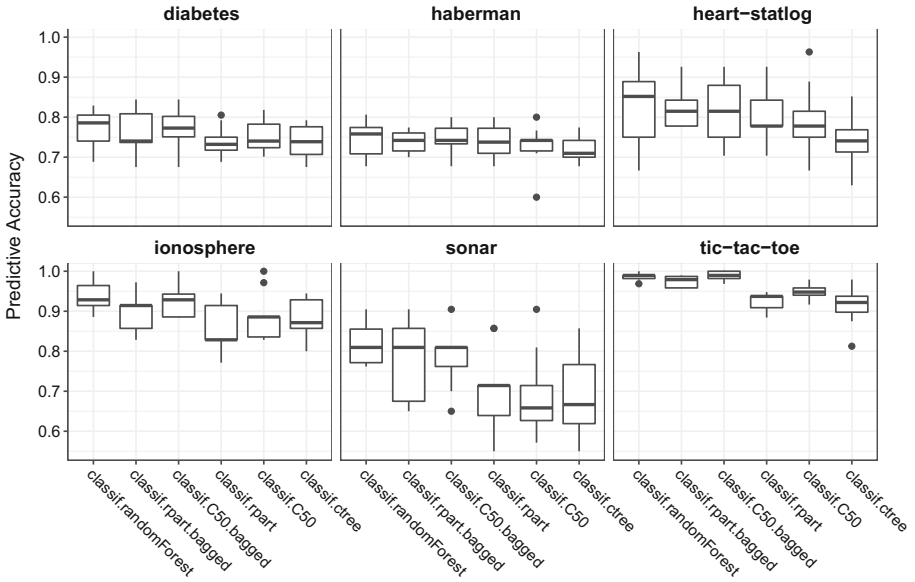



**Fig. 3** Cross-validated predictive accuracy per learner and task. Each *boxplot* contains 10 values for one complete cross-validation

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

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

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

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

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

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

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

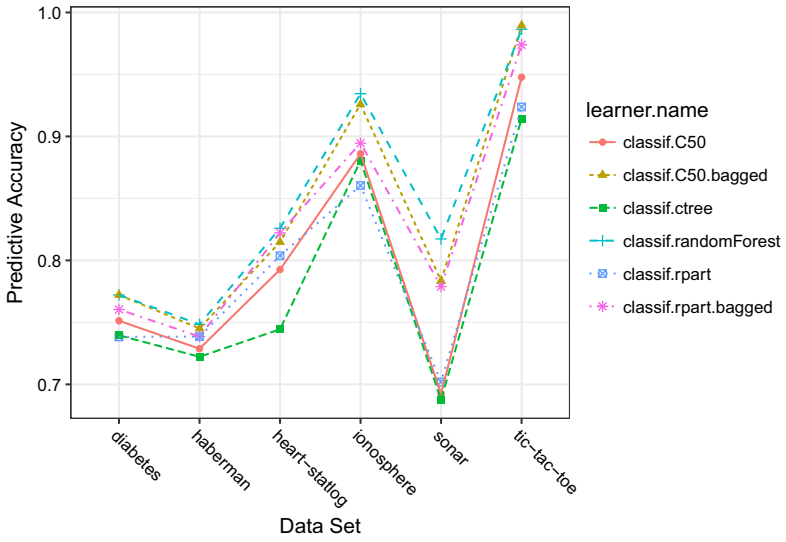



**Fig. 4** Results of the produced runs. Each *point* represents the averaged predictive accuracy over all cross-validation iterations generated by running a particular learner on the respective task

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

## 6 Conclusion and outlook

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

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

## References

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

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

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

# 5. OpenML Benchmarking Suites

Chapter 5 promotes the use of benchmarking suites (i.e., a carefully selected collection of easily accessible datasets). The chapter also describes how researchers can create their own benchmarking suites on OpenML, and it presents a first such collection for classification datasets, namely the *OpenML100* benchmarking suite.

**Contributing article:**

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2017). OpenML Benchmarking Suites and the OpenML100. *arXiv preprint arXiv:1708.03731*. `https://arxiv.org/abs/1708.03731`. In Preparation.

**Author contributions:**

The paper was jointly written and reworked by all authors. All authors contributed to the extensive task of selecting, documenting, and curating appropriate classification datasets from OpenML. Jan van Rijn and Joaquin Vanschoren were responsible for extending the REST API, and they also implemented an interface to the OpenML server for creating benchmarking suites. Matthias Feurer and Jan van Rijn were responsible for extending and adapting the Python interface. Giuseppe Casalicchio was responsible for extending and adapting the R interface. All authors added valuable input and proofread the paper in several stages.

OpenML100

# OpenML Benchmarking Suites and the OpenML100

**Bernd Bischl** bernd.bischl@stat.uni-muenchen.de
**Giuseppe Casalicchio** giuseppe.casalicchio@stat.uni-muenchen.de
**Matthias Feurer** feurerm@cs.uni-freiburg.de
**Frank Hutter** fh@cs.uni-freiburg.de
**Michel Lang** lang@statistik.tu-dortmund.de
**Rafael G. Mantovani** rgmantov@icmc.usp.br
**Jan N. van Rijn** vanrijn@cs.uni-freiburg.de
**Joaquin Vanschoren** j.vanschoren@tue.nl

**Editor:** To be determined

## Abstract

We advocate the use of curated, comprehensive benchmark suites of machine learning datasets, backed by standardized OpenML-based interfaces and complementary software toolkits written in Python, Java and R. Major distinguishing features of OpenML benchmark suites are (a) ease of use through standardized data formats, APIs, and existing client libraries; (b) machine-readable meta-information regarding the contents of the suite; and (c) online sharing of results, enabling large scale comparisons. As a first such suite, we propose the OpenML100, a machine learning benchmark suite of 100 classification datasets carefully curated from the thousands of datasets available on OpenML.org.

**Keywords:** machine learning, benchmarking

## 1. A Brief History of Benchmarking Suites

Proper algorithm benchmarking is a hallmark of machine learning research. It allows us, as a community, to track progress over time, to identify still challenging issues, and to learn which algorithms are most appropriate for specific applications. However, we currently lack standardized, easily-accessible benchmark suites of datasets that are curated to reflect important problem domains, practical to use, and that support a rigorous analysis of performance results. This often results in suboptimal shortcuts in study designs, producing rather small-scale, one-off experiments that should be interpreted with caution (Aha, 1992), are hard to reproduce (Pedersen, 2008; Hirsh, 2008), and may even lead to contradictory results (Keogh and Kasetty, 2003).

The machine learning field has long recognized the importance of dataset repositories. The UCI repository (Lichman, 2013) offers a wide range of datasets, but it does not attempt to make them available through a uniform format or API. The same holds for other repositories, such as LIBSVM (Chang and Lin, 2011). `mldata.org` is a very popular repository that does provide an API to easily download datasets, and is readily integrated in scikit-learn. However, it is no longer being maintained, and will very likely be merged with

our OpenML. KEEL (Alcala et al., 2010) offers some benchmark data suites, including one for imbalanced classification and one with data sets with missing values. It has a Java toolkit and an R library for convenient access. Likewise, PMLB (Olson et al., 2017) is another collection of datasets, with strong overlap to UCI, with tools to import them into Python scripts. However, none of the above tools allows to add new datasets or easily share and compare benchmarking results online.[1] Other related benchmark collections include UCR (Chen et al., 2015) for time series data, OpenAI gym (Brockman et al., 2016) for reinforcement learning problems, and Mulan (Tsoumakas et al., 2011) for multilabel datasets, with some of the multilabel datasets already available on OpenML (Probst et al., 2017).

All of these existing repositories are rather well-curated, and for many years machine learning researchers have benchmarked their algorithms on a subset of their data sets. However, most of them do not provide APIs for downloading data in standardized formats into popular machine learning libraries and uploading and comparing the ensuing results. Hence, large scale benchmarks that also build upon previous results of others are still the exception.

## 2. OpenML Benchmarking Suites

We advocate expanding on previous efforts by comprehensive benchmark suites backed by the open machine learning platform OpenML (Vanschoren et al., 2013). Our goal is to substantially facilitate in-depth benchmarking by providing a standard set of datasets covering a wide spectrum of domains and statistical properties, together with rich meta-data and standardized evaluation procedures (i.e., we also provide unified data splits for resampling methods). This eliminates guesswork, makes individual results more comparable, and allows more standardized analysis of all results. In addition, we provide software libraries in several programming languages to easily download these datasets, optionally download prior benchmarking results for reuse and comparison, and to share your results online.

OpenML is an online platform for reproducible, collaborative machine learning experiments and can be used to store and share all aspects of machine learning experiments, including data, code, experiment parameters and results. All our datasets in OpenML are provided in a uniform format, highlight issues such as unique-valued or constant features, include extensive meta-data for deeper analysis of evaluation results, and provide task-specific meta-data, such as target features and predefined train-test splits.

Researchers can conveniently explore the datasets included in OpenML through comprehensive APIs to find suitable learning tasks for their planned experiments, depending on required data set characteristics. These APIs allow, for instance, to find all high-dimensional data sets with few observations and no missing values.

## 3. The OpenML100 Benchmarking Suite

On top of OpenML's customizable functionality, we provide a new standard benchmark suite of 100 high-quality datasets carefully curated from the many thousands available on OpenML: the OpenML100.

1. The latter used to be possible with DELVE (`http://www.cs.toronto.edu/~delve/`) and `mlcomp.org`, but both services are no longer maintained.

OpenML100

We selected classification datasets for this benchmarking suite to satisfy the following requirements: (a) the number of observations are between 500 and 100 000 to focus on medium-sized datasets, (b) the number of features does not exceed 5000 features to keep the runtime of algorithms low, (c) the target attribute has at least two classes, and (d) the ratio of the minority class and the majority class is above 0.05 (to eliminate highly imbalanced datasets). We excluded datasets which (a) cannot be randomized via a 10-fold cross-validation due to grouped samples, (b) are a subset of a larger dataset available on OpenML, (c) have no source or reference available, (d) are created by binarization of regression tasks or multiclass classification tasks, or (e) include sparse data (e.g., text mining data sets). A detailed list of the data properties can be found on OpenML[2].

## 4. How to use the OpenML100

In this section we demonstrate how our dataset collection can be conveniently imported for benchmarking using our client libraries in Python, Java and R. Figure 1 provides exemplary code chunks for downloading the datasets and running a basic classifier in all three languages. In these examples, we use the Python library with scikit-learn (Pedregosa et al., 2011), the R package (Casalicchio et al., 2017) with mlr (Bischl et al., 2016), and the Java library with Weka (Hall et al., 2009). OpenML has also been integrated in MOA and RapidMiner (van Rijn and Vanschoren, 2015).

OpenML works with the concept of *tasks* to facilitate comparable and reproducible results. A task extends a dataset with task-specific information, such as target attributes and evaluation procedures. Datasets and tasks are automatically downloaded at first use and are afterwards cached locally. *Studies* combine a specific set of tasks and can also hold all benchmarking results obtained on them. In the code examples, the OpenML100 tasks are downloaded through the study with the same name. They also show how to access the raw data set (although this is not needed to train a model), fit a simple classifier on the defined data splits, and finally publish runs on the OpenML server. Note that the Java implementation automatically uploads results to the server.

## 5. Creating new Benchmarking Suites

The set of datasets on `OpenML.org` can easily be extended, and additional OpenML benchmark suites, e.g., for regression and time-series data, can easily be created by defining sets of datasets according to specific needs. Instructions for creating new benchmarking suites can be found on `https://www.openml.org`. We currently envision two routes of extensions: (a) facilitate the creation and versioning of these benchmark suites on `OpenML.org`; and (b) adding automatic statistical analysis, visualization and reporting on the online platform.

**Acknowledgements** This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721, through grant #2015/03986-0 from the São Paulo Research Foundation (FAPESP), as well as Priority Programme Autonomous Learning (SPP 1527, grant

2. `https://www.openml.org/s/14`

```python
import openml
import sklearn
benchmark_suite = openml.study.get_study('OpenML100','tasks') # obtain the benchmark suite
clf = sklearn.pipeline.Pipeline(steps=[('imputer',sklearn.preprocessing.Imputer()), ('estimator',
  sklearn.tree.DecisionTreeClassifier())])              # build a sklearn classifier
for task_id in benchmark_suite.tasks:                    # iterate over all tasks
  task = openml.tasks.get_task(task_id)                  # download the OpenML task
  X, y = task.get_X_and_y()                              # get the data (not used in this example)
  openml.config.apikey = 'FILL_IN_OPENML_API_KEY'        # set the OpenML Api Key
  run = openml.runs.run_model_on_task(task,clf)          # run classifier on splits (requires API key)
  score = run.get_metric_score(sklearn.metrics.accuracy_score) # print accuracy score
  print('Data set: %s; Accuracy: %0.2f' % (task.get_dataset().name,score.mean()))
  run.publish()                                          # publish the experiment on OpenML (optional)
  print('URL for run: %s/run/%d' %(openml.config.server,run.run_id))
```

(a) Python, available on https://github.com/openml/openml-python/

```java
public static void runTasksAndUpload() throws Exception {
  OpenmlConnector openml = new OpenmlConnector();
  Study benchmarksuite = openml.studyGet("OpenML100", "tasks");      // obtain the benchmark suite
  Classifier tree = new REPTree();                                   // build a Weka classifier
  for (Integer taskId : benchmarksuite.getTasks()) {                 // iterate over all tasks
    Task t = openml.taskGet(taskId);                                 // download the OpenML task
    Instances d = InstancesHelper.getDatasetFromTask(openml, t);     // obtain the dataset
    openml.setApiKey("FILL_IN_OPENML_API_KEY");
    int runId = RunOpenmlJob.executeTask(openml, new WekaConfig(), taskId, tree);
    Run run = openml.runGet(runId);}}                                // retrieve the uploaded run
```

(b) Java, available on Maven Central with artifact id 'org.openml.openmlweka'

```r
library(OpenML)
lrn = makeLearner('classif.rpart')               # construct a simple CART classifier
task.ids = getOMLStudy('OpenML100')$tasks$task.id # obtain the list of suggested tasks
for (task.id in task.ids) {                      # iterate over all tasks
  task = getOMLTask(task.id)                     # download single OML task
  data = as.data.frame(task)                     # obtain raw data set
  run = runTaskMlr(task, learner = lrn)          # run constructed learner
  setOMLConfig(apikey = 'FILL_IN_OPENML_API_KEY')
  upload = uploadOMLRun(run)                     # upload and tag the run
}
```

(c) R, available on CRAN via package OpenML

Figure 1: Running classifiers on a task and (optionally) uploading the results. Uploading requires the user to fill in an API key.

HU 1900/3-1) and Collaborative Research Center SFB 876/A3 from the German Research Foundation (DFG).

## References


D. W. Aha. Generalizing from case studies: A case study. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–10, 1992.

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

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

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

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

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

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

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.

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

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.

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

R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: A large benchmark suite for machine learning evaluation and comparison. *arXiv:1703.00512*, 2017.

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

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.

P. Probst, Q. Au, G. Casalicchio, C. Stachl, and B. Bischl. Multilabel classification with R package mlr. *The R Journal*, 9(1):352–369, 2017.

G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *JMLR*, pages 2411–2414, Jul 2011.

J. N. van Rijn and J. Vanschoren. Sharing RapidMiner workflows and experiments with OpenML. In *MetaSel@ PKDD/ECML*, pages 93–103, 2015.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

## Appendix A. Datasets Included in the OpenML100 Benchmark Suite

Table 1: Datasets included in the OpenML100 benchmark suite. For each dataset, we show: the OpenML task id and name, the number of classes (nClass), features (nFeat) and observations (nObs), as well as the ratio of the minority and majority class sizes (ratioMinMaj).

| Task id | Name | nClass | nFeat | nObs | ratioMinMaj |
|---|---|---|---|---|---|
| 3 | kr-vs-kp | 2 | 37 | 3196 | 0.91 |
| 6 | letter | 26 | 17 | 20000 | 0.90 |
| 11 | balance-scale | 3 | 5 | 625 | 0.17 |
| 12 | mfeat-factors | 10 | 217 | 2000 | 1.00 |
| 14 | mfeat-fourier | 10 | 77 | 2000 | 1.00 |
| 15 | breast-w | 2 | 10 | 699 | 0.53 |
| 16 | mfeat-karhunen | 10 | 65 | 2000 | 1.00 |
| 18 | mfeat-morphological | 10 | 7 | 2000 | 1.00 |
| 20 | mfeat-pixel | 10 | 241 | 2000 | 1.00 |
| 21 | car | 4 | 7 | 1728 | 0.05 |
| 22 | mfeat-zernike | 10 | 48 | 2000 | 1.00 |
| 23 | cmc | 3 | 10 | 1473 | 0.53 |
| 24 | mushroom | 2 | 23 | 8124 | 0.93 |
| 28 | optdigits | 10 | 65 | 5620 | 0.97 |
| 29 | credit-a | 2 | 16 | 690 | 0.80 |
| 31 | credit-g | 2 | 21 | 1000 | 0.43 |
| 32 | pendigits | 10 | 17 | 10992 | 0.92 |
| 36 | segment | 7 | 20 | 2310 | 1.00 |
| 37 | diabetes | 2 | 9 | 768 | 0.54 |
| 41 | soybean | 19 | 36 | 683 | 0.09 |
| 43 | spambase | 2 | 58 | 4601 | 0.65 |
| 45 | splice | 3 | 62 | 3190 | 0.46 |
| 49 | tic-tac-toe | 2 | 10 | 958 | 0.53 |
| 53 | vehicle | 4 | 19 | 846 | 0.91 |
| 58 | waveform-5000 | 3 | 41 | 5000 | 0.98 |
| 219 | electricity | 2 | 9 | 45312 | 0.74 |
| 2074 | satimage | 6 | 37 | 6430 | 0.41 |
| 2079 | eucalyptus | 5 | 20 | 736 | 0.49 |
| 3021 | sick | 2 | 30 | 3772 | 0.07 |
| 3022 | vowel | 11 | 13 | 990 | 1.00 |
| 3481 | isolet | 26 | 618 | 7797 | 0.99 |
| 3485 | scene | 2 | 300 | 2407 | 0.22 |
| 3492 | monks-problems-1 | 2 | 7 | 556 | 1.00 |
| 3493 | monks-problems-2 | 2 | 7 | 601 | 0.52 |
| 3494 | monks-problems-3 | 2 | 7 | 554 | 0.92 |
| 3510 | JapaneseVowels | 9 | 15 | 9961 | 0.48 |
| 3512 | synthetic_control | 6 | 62 | 600 | 1.00 |
| 3543 | irish | 2 | 6 | 500 | 0.80 |
| 3549 | analcatdata_authorship | 4 | 71 | 841 | 0.17 |
| 3560 | analcatdata_dmft | 6 | 5 | 797 | 0.79 |
| 3561 | profb | 2 | 10 | 672 | 0.50 |
| 3567 | collins | 15 | 24 | 500 | 0.07 |
| 3573 | mnist_784 | 10 | 785 | 70000 | 0.80 |
| 3889 | sylva_agnostic | 2 | 217 | 14395 | 0.07 |
| 3891 | gina_agnostic | 2 | 971 | 3468 | 0.97 |
| 3896 | ada_agnostic | 2 | 49 | 4562 | 0.33 |
| 3899 | mozilla4 | 2 | 6 | 15545 | 0.49 |
| 3902 | pc4 | 2 | 38 | 1458 | 0.14 |
| 3903 | pc3 | 2 | 38 | 1563 | 0.11 |
| 3904 | jm1 | 2 | 22 | 10885 | 0.24 |
| 3913 | kc2 | 2 | 22 | 522 | 0.26 |
| 3917 | kc1 | 2 | 22 | 2109 | 0.18 |
| 3918 | pc1 | 2 | 22 | 1109 | 0.07 |
| 3946 | KDDCup09_churn | 2 | 231 | 50000 | 0.08 |
| 3948 | KDDCup09_upselling | 2 | 231 | 50000 | 0.08 |
| 3950 | musk | 2 | 170 | 6598 | 0.18 |
| 3954 | MagicTelescope | 2 | 12 | 19020 | 0.54 |
| 7592 | adult | 2 | 15 | 48842 | 0.31 |
| 9914 | wilt | 2 | 6 | 4839 | 0.06 |
| 9946 | wdbc | 2 | 31 | 569 | 0.59 |
| 9950 | micro-mass | 20 | 1301 | 571 | 0.18 |
| 9952 | phoneme | 2 | 6 | 5404 | 0.42 |
| 9954 | one-hundred-plants-margin | 100 | 65 | 1600 | 1.00 |
| 9955 | one-hundred-plants-shape | 100 | 65 | 1600 | 1.00 |
| 9956 | one-hundred-plants-texture | 100 | 65 | 1599 | 0.94 |
| 9957 | qsar-biodeg | 2 | 42 | 1055 | 0.51 |
| 9960 | wall-robot-navigation | 4 | 25 | 5456 | 0.15 |
| 9964 | semeion | 10 | 257 | 1593 | 0.96 |
| 9967 | steel-plates-fault | 2 | 34 | 1941 | 0.53 |
| 9968 | tamilnadu-electricity | 20 | 4 | 45781 | 0.48 |
| 9970 | hill-valley | 2 | 101 | 1212 | 1.00 |
| 9971 | ilpd | 2 | 11 | 583 | 0.40 |
| 9976 | madelon | 2 | 501 | 2600 | 1.00 |
| 9977 | nomao | 2 | 119 | 34465 | 0.40 |
| 9978 | ozone-level-8hr | 2 | 73 | 2534 | 0.07 |
| 9979 | cardiotocography | 10 | 36 | 2126 | 0.09 |
| 9980 | climate-model-simulation-crashes | 2 | 21 | 540 | 0.09 |
| 9981 | cnae-9 | 9 | 857 | 1080 | 1.00 |
| 9983 | eeg-eye-state | 2 | 15 | 14980 | 0.81 |
| 9985 | first-order-theorem-proving | 6 | 52 | 6118 | 0.19 |
| 9986 | gas-drift | 6 | 129 | 13910 | 0.55 |
| 10093 | banknote-authentication | 2 | 5 | 1372 | 0.80 |
| 10101 | blood-transfusion-service-center | 2 | 5 | 748 | 0.31 |
| 14964 | artificial-characters | 10 | 8 | 10218 | 0.42 |
| 14965 | bank-marketing | 2 | 17 | 45211 | 0.13 |
| 14966 | Bioresponse | 2 | 1777 | 3751 | 0.84 |
| 14967 | cjs | 6 | 35 | 2796 | 0.40 |
| 14968 | cylinder-bands | 2 | 40 | 540 | 0.73 |
| 14969 | GesturePhaseSegmentationProcessed | 5 | 33 | 9873 | 0.34 |
| 14970 | har | 6 | 562 | 10299 | 0.72 |
| 34536 | Internet-Advertisements | 2 | 1559 | 3279 | 0.16 |
| 34537 | PhishingWebsites | 2 | 31 | 11055 | 0.80 |
| 34538 | MiceProtein | 8 | 82 | 1080 | 0.70 |
| 34539 | Amazon_employee_access | 2 | 10 | 32769 | 0.06 |
| 125920 | dresses-sales | 2 | 13 | 500 | 0.72 |
| 125921 | LED-display-domain-7digit | 10 | 8 | 500 | 0.65 |
| 125922 | texture | 11 | 41 | 5500 | 1.00 |
| 125923 | Australian | 2 | 15 | 690 | 0.80 |
| 146606 | higgs | 2 | 29 | 98050 | 0.89 |
| 146607 | SpeedDating | 2 | 123 | 8378 | 0.20 |

# Part III.

# Evaluation and Interpretation of Machine Learning Models

# 6. The Residual-Based Predictiveness Curve

The predictiveness curve (Huang et al., 2007) is a visualization method to assess the performance of binary classifiers that predict probabilities. Regarding the assessment of such probabilistic classifiers, two essential criteria must be considered, namely the discrimination performance and the calibration of the predicted probabilities. The ROC curve is known to consider only the discrimination performance and not the calibration of the predicted probabilities. Therefore, the use of the predictiveness curve is a reasonable alternative, since it also accounts for the calibration of the predicted probabilities (cf. Pepe et al., 2008). Chapter 6 reviews the role of the predictiveness curve in the performance assessment and discusses several shortcomings of the curve. Furthermore, the RBP curve is proposed as an extension that addresses several shortcomings of the original predictiveness curve. This chapter also shows how the RBP curve can be used to derive several discrimination and calibration measures.

**Contributing article:**

Casalicchio, G., Bischl, B., Boulesteix, A.-L., and Schmid, M. (2016). The Residual-based Predictiveness Curve: A Visual Tool to Assess the Performance of Prediction Models. *Biometrics*, 72(2):392–401. *https://doi.org/10.1111/biom.12455*.

**Copyright information:**



**Author contributions:**

The whole paper was drafted and, in most parts, written by Giuseppe Casalicchio. Matthias Schmid revised the paper in several stages and contributed to all sections. Bernd Bischl helped with setting up the simulation and designing the accompanying R package. Anne-Laure Boulesteix contributed to the real data application in Section 5. All authors added valuable input, proofread and revised the paper.

**Supplementary material available at:**

- R package: `https://cran.r-project.org/web/packages/RBPcurve`
- Supplementary material: `https://onlinelibrary.wiley.com/doi/abs/10.1111/biom.12455`

Biometrics 72, 392–401
June 2016

DOI: 10.1111/biom.12455

# The Residual-Based Predictiveness Curve: A Visual Tool to Assess the Performance of Prediction Models

**Giuseppe Casalicchio,[1,*] Bernd Bischl,[1,**] Anne-Laure Boulesteix,[2,***] and Matthias Schmid[3,****]**

[1]Department of Statistics, University of Munich, 80539 Munich, Germany
[2]Department of Medical Informatics, Biometry and Epidemiology, University of Munich, 81377 Munich, Germany
[3]Department of Medical Biometry, Informatics and Epidemiology, University of Bonn, 53105 Bonn, Germany
*email: giuseppe.casalicchio@stat.uni-muenchen.de
**email: bernd.bischl@stat.uni-muenchen.de
***email: boulesteix@ibe.med.uni-muenchen.de
****email: matthias.schmid@ukb.uni-bonn.de

Summary. It is agreed among biostatisticians that prediction models for binary outcomes should satisfy two essential criteria: first, a prediction model should have a high discriminatory power, implying that it is able to clearly separate cases from controls. Second, the model should be well calibrated, meaning that the predicted risks should closely agree with the relative frequencies observed in the data. The focus of this work is on the predictiveness curve, which has been proposed by Huang et al. (Biometrics 63, 2007) as a graphical tool to assess the aforementioned criteria. By conducting a detailed analysis of its properties, we review the role of the predictiveness curve in the performance assessment of biomedical prediction models. In particular, we demonstrate that marker comparisons should not be based solely on the predictiveness curve, as it is not possible to consistently visualize the added predictive value of a new marker by comparing the predictiveness curves obtained from competing models. Based on our analysis, we propose the "residual-based predictiveness curve" (RBP curve), which addresses the aforementioned issue and which extends the original method to settings where the evaluation of a prediction model on independent test data is of particular interest. Similar to the predictiveness curve, the RBP curve reflects both the calibration and the discriminatory power of a prediction model. In addition, the curve can be conveniently used to conduct valid performance checks and marker comparisons.

Key words: Calibration; Classification; Discrimination; Predictiveness curve; Risk prediction.

## 1. Introduction

The development of prediction models for binary outcomes is an important issue in biomedical research (Moons et al., 2009). Key issues in biostatistical method development are not only to propose methods for *deriving* new marker combinations but also to develop reliable methods to *evaluate* and *assess* the predictive value of a new model.

In the biomedical sciences, a binary outcome $D$ often refers to the status of a disease, with diseased ($D = 1$) and healthy ($D = 0$) subjects being considered cases and controls, respectively. It is commonly agreed that prediction models for a binary outcome should satisfy two major criteria: first, they should have a high *discriminatory* power, meaning that they are able to separate the two categories of the outcome. Second, they should be *well calibrated*, meaning that the predicted risks should closely agree with the relative frequencies observed in the data. Both discrimination and calibration can be evaluated by a variety of measures (see Section 2).

The focus of this article is on the use of the predictiveness curve (Huang, Pepe, and Feng, 2007), which is a popular graphical tool to evaluate and compare the performance of prediction models (see Johnson et al., 2010; Soto et al., 2013 and Web Appendix A for further references on this topic). The predictiveness curve depicts the risk distribution of a marker (or a marker combination) $Y$ and is formally defined as follows: Let $F$ be the cumulative distribution function (cdf) of $Y$, and let $\nu = F(Y) \in (0, 1)$ be the $\nu$-th percentile of $Y$. The risk at a specific value $\nu$ is then given by $R(\nu) = risk(F^{-1}(\nu)) := \mathbb{P}(D = 1 \mid Y = F^{-1}(\nu))$, and the predictiveness curve is obtained by plotting $R(\nu)$ versus $\nu$.

In situations where $R(\cdot)$ is not known, statistical estimation of the risk is required. This is typically done by fitting a prediction model to an i.i.d. data set $\mathcal{D} = \{(\mathbf{x}_i, D_i), i = 1, \ldots, n\}$ with $n$ observations, where $D_i$ denotes the disease status of subject $i$ and $\mathbf{x}_i = (1, x_{i,1}, \ldots, x_{i,P})^\top$ is the associated vector of $P$ marker values that may include the constant 1 for the intercept term. A prediction model is then obtained by specifying, for example, a score $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$ via

$$risk(\eta_i) = \mathbb{P}(D = 1 \mid Y = \eta_i) = G(\eta_i), \quad i = 1, \ldots, n, \qquad (1)$$

where $G$ is a pre-defined monotone increasing function that transforms the range of the score to the interval $(0, 1)$ and where $\boldsymbol{\beta}$ is a vector of unknown coefficients that has to be estimated. This parametrization is widely used in parametric modeling, for example in logistic regression. An estimate of the predictiveness curve is then obtained by plotting the predicted risks obtained from the prediction model

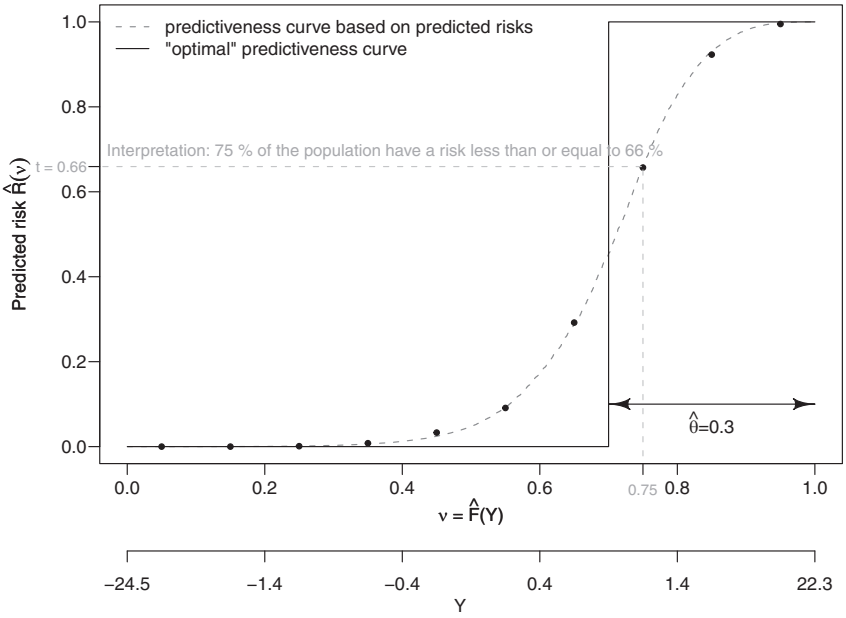



**Figure 1.** Predictiveness curve obtained from the simulated data set of Section 1 with estimated prevalence $\hat{\theta} = 0.3$. The dashed gray line represents the predictiveness curve of a logistic regression model while the solid black line corresponds to the optimal predictiveness curve that was obtained by using the binary outcome values. The black points correspond to the proportion of diseased within each decile of predicted risks. Because the black points are close to the respective predictiveness curve, they indicate that the logistic regression model is well calibrated.

$\hat{R}(\nu) = \widehat{risk}(\hat{F}^{-1}(\nu))$ versus $\nu = \hat{F}(\hat{\eta})$, where $\hat{\eta}$ refers to the estimated linear predictor $\hat{\eta} = \mathbf{x}^\top \hat{\boldsymbol{\beta}}$ and $\hat{F}$ is the empirical c.d.f. of $Y \equiv \hat{\eta}$ (Gu and Pepe, 2009). To unify notation, we also allow $\hat{\eta}$ to refer to the observed value of a single marker $Y$, although no estimation is involved in this case. An even more general situation than (1) is given when $\widehat{risk}(\cdot)$ refers to a probability estimate obtained from some arbitrary statistical estimation technique. This situation includes the special case where the marker $Y$ itself is defined as a probability estimate, i.e., $Y \equiv \widehat{risk}(Y)$. For a given i.i.d. data set, the prevalence $\theta = P(D = 1)$ can be estimated by $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} D_i$.

Figure 1 shows the predictiveness curve obtained from a simulated data set with $n = 10,000$ observations and $P = 5$ standard normally distributed markers $X_1, \ldots, X_5 \sim N(0, 1)$. The values of the disease status $D$ were generated from the model $\text{logit}(P(D = 1 \mid \mathbf{x}_i)) = \mathbf{x}_i^\top \boldsymbol{\beta}$ with randomly generated coefficients $\boldsymbol{\beta} = (-2.99, -2.53, -2.04, -2.35, -2.17, -2.75)^\top$. The first coefficient corresponds to the intercept and the others correspond to the five markers. It is seen that the predictiveness curve characterizes the distribution of the risk: each horizontal line at $\hat{R}(\nu) = t$ intersects the curve at a specific abscissa $\nu$, where $\nu = \hat{R}^{-1}(t)$ is interpreted as the proportion of the population that has a risk of being a case less than or equal to $t$. The estimated prevalence in the data was $\hat{\theta} = 0.3$. If the prediction model is well calibrated, the step function that jumps up from 0 to 1 at $\nu = 1 - \hat{\theta}$ can be considered as the "optimal" predictiveness curve that would be obtained from a model with perfect discriminatory power. Consequently, a steep slope of the curve indicates that cases and controls are well separated, and a predictiveness curve that is close to the "optimal" curve is interpreted as having a high discriminatory power (Huang et al., 2007).

In this article, we carry out a detailed analysis of the properties of the predictiveness curve and review its role in the performance assessment of prediction models. A key result is that the predictiveness curve should not be used as a criterion for marker or model comparison, unless one has made sure that all prediction models under consideration are well calibrated. This result is in line with Cook (2010), Pepe (2010), and Pepe et al. (2013), who suggested first checking whether all prediction models under consideration are well calibrated *before* comparing the predictiveness curves with regard to their slopes.

We further propose a new graphical tool, the residual-based predictiveness curve, hereinafter abbreviated as *RBP curve*, which does not depend on the results of previously conducted calibration checks. The idea of the RBP curve is to incorporate the binary outcome values into the definition of the predictiveness curve, thereby defining a new evaluation criterion for marker performance that also reveals how well a model is calibrated. The properties of the RBP curve will be assessed in detail in Section 3, and it will be demonstrated that several well-known performance criteria, such as the classification accuracy, the true positive rate (TPR) and the false positive rate (FPR) can be derived graphically from the RBP curve.

## 2. Calibration and Discrimination

An important issue in the assessment of marker performance is to quantify *calibration* and *discrimination*, i.e., to define appropriate measures for evaluating the two criteria. In this respect, Pepe et al. (2008) and Gu and Pepe (2009) have shown that many relevant properties of a marker can be summarized through the predictiveness curve. The connections between the predictiveness curve and measures of calibration and discrimination are described in the following subsections.

### 2.1. *Measures of Calibration*

Pepe et al. (2013) distinguish between *good calibration* of a model and a *well calibrated* model. A measure for *good calibration* is given by the *calibration-in-the-large*, which measures how well the average risk over all observations agrees with the proportion of diseased subjects. *Good calibration* is satisfied when $\mathbb{E}(risk(Y))$, which is estimated by the mean risk $\frac{1}{n}\sum_{i=1}^{n}\widehat{risk}(\hat{\eta}_i)$, is close to the prevalence $\theta$, estimated by $\hat{\theta}=\frac{1}{n}\sum_{i=1}^{n}D_i$. Therefore, *good calibration* is obtained when the area under the predictiveness curve is close to the area under the "optimal" predictiveness curve, which is the area of a rectangle with width $\hat{\theta}$ and height 1 (see Figure 1). Note that these considerations require $\hat{\theta}$ to be a valid estimate of the prevalence (which is, e.g., the case if the data are a random sample from the underlying population).

Cook (2010) pointed out that evaluating only the calibration-in-the-large is not sufficient to fully assess the calibration of a prediction model. Rather, it should be seen as the most basic requirement because many aspects of calibration may not be accounted for even if calibration-in-the-large is satisfied. A more precise measure for a *well calibrated* model can be obtained by grouping the observations via quantiles of the predicted risks and by comparing the mean of the predicted risk in each group to the respective proportion of diseased subjects within each group. This strategy gives rise to the Hosmer–Lemeshow statistic (e.g., Crowson et al., 2014). Pepe et al. (2008) suggested visualizing the components of the Hosmer–Lemeshow statistic by additionally plotting the observed proportions of diseased subjects at the midpoint of each decile of the predicted risks (see Figure 1). If the points are close to the predictiveness curve, the prediction model in question is likely to be *well calibrated*.

### 2.2. *Measures of Discrimination*

The discriminatory power of a *well calibrated* prediction model can be visually assessed by inspecting the slope of the predictiveness curve. Because observations with higher risks are located to the right side of the vertical line at $v=1-\hat{\theta}$, a well discriminating model should assign risk values close to 1 to these observations. Conversely, it should assign risk values close to 0 to the observations to the left side, i.e., at $v<1-\hat{\theta}$.

A popular measure to quantify the discriminatory power of a prediction model is the mean risk difference $\text{MRD} := \mathbb{E}(risk(Y)\,|\,D=1) - \mathbb{E}(risk(Y)\,|\,D=0)$, which is also known as discrimination slope and which is closely related to the proportion of explained variation ("PEV", Pepe, Feng, and Gu, 2008; Gu and Pepe, 2009). MRD is the difference between the conditional expectations of $risk(Y)$ in the diseased and healthy groups. Because the first expectation should be close to 1 and the second one close to 0 in a well discriminating model, the MRD is directly related to the slope of the predictiveness curve. The steeper the slope, the better the population is separated into two groups (Huang et al., 2007). If, at the same time, the model is *well calibrated*, a steep slope of the curve implies a large value of MRD. Empirically, the measure can be estimated by $\widehat{\text{MRD}}=\frac{1}{n_1}\sum_{i:D_i=1}G(\hat{\eta}_i)-\frac{1}{n_0}\sum_{i:D_i=0}G(\hat{\eta}_i)$, where $G(\hat{\eta}_i)$ is the parametrization of the estimated risk similar to equation (1) and $n_j:=\sum_{i=1}^{n}I(D_i=j)$, $j=0,1$, are the numbers of healthy and diseased subjects, respectively.

## 3. The RBP Curve

### 3.1. *Definition and Derivation of the RBP Curve*

Although the predictiveness curve has proved to be a valuable tool for the characterization of biomarker combinations, several shortcomings regarding the evaluation of prediction models remain. In particular, it can be misleading to rely on the predictiveness curve when the aim is to *compare* different marker combinations. This problem is mainly due to the fact that the predictiveness curve does not involve any of the true values of the outcome variable $D$. It is, therefore, possible to construct algorithms that "optimize" marker combinations such that they result in the "optimal" step function of Figure 1 although they are impractical and badly calibrated (see Web Appendix B, where we derive and illustrate such an algorithm).

Another problem, which will be discussed in more detail in Section 4.1, is that the predictiveness curve cannot be used to evaluate whether a prediction model overfitted the data it was derived from. Evaluating the curve on external test data does not help in this respect, as the predictiveness curve only displays the distribution of $risk(Y)$. Consequently, if the covariates in the test data follow the same distribution as those in the original data, both curves will be highly similar regardless of whether the model overfits the original data or not.

As noted in Section 1, several authors have suggested conducting marker comparisons only if the prediction models under consideration are *well calibrated*, which is achieved by taking into account the real outcome values. Although this strategy solves part of the problem, it is highly dependent on how well the calibration check works. In particular, wrong conclusions are possible if low-power tests such as the Hosmer–Lemeshow test are used (Cook, 2010).

To address these issues, we propose using the residuals $\epsilon = D - risk(Y)$ and plot $\epsilon$ versus the c.d.f. of the residuals $F_\epsilon(\epsilon)$. Because the real outcome $D$ is included in the definition of $\epsilon$, this strategy allows one to directly assess the calibration of a prediction model without plotting any additional points as in Figure 1. When statistical estimation is involved (as described in Section 1), we consider the empirical c.d.f. of the residuals denoted by $\hat{F}_\epsilon(\hat{\epsilon})$ and the estimated residuals $\hat{\epsilon}_i = D_i - \hat{p}_i$, where $\hat{p}_i := \widehat{risk}(\hat{\eta}_i) \in [0,1]$ and $D_i \in \{0,1\}$. This means that in the first step, a probability model for $\widehat{risk}(Y)$ is used to transform the values of $Y$ to the interval $[0,1]$ and to calculate the estimated residuals $\hat{\epsilon} = D - \widehat{risk}(Y)$. In the next step, the empirical c.d.f. of the residuals, denoted by $\hat{F}_\epsilon(\hat{\epsilon})$, is used to plot $\hat{\epsilon}$ versus $\hat{F}_\epsilon(\hat{\epsilon})$. Note that the RBP curve can

be applied whenever the original predictiveness curve can be applied; it only requires the existence of a probability model to calculate $\widehat{risk}(Y)$.

Figure 2 illustrates the RBP curve and how various criteria (calibration-in-the-large, calibration across deciles, MRD, classification accuracy, FPR, and TPR) can be obtained from it. The horizontal line corresponds to the optimal RBP curve where all residuals are zero. By definition, all non-diseased subjects $\{i : D_i = 0\}$ are located below the horizontal zero line, as $\hat{\epsilon}_i = D_i - \hat{p}_i = -\hat{p}_i \leq 0$. Accordingly, all diseased subjects $\{i : D_i = 1\}$ are located above the horizontal line. The vertical line that splits diseased and non-diseased subjects is located at $1 - \hat{\theta}$, i.e., at one minus the prevalence. Therefore one can derive the proportion of diseased subjects from this line.

### 3.2. *Relation between the RBP Curve and Other Performance Measures*

*Relation to the calibration-in-the-large.* The calibration-in-the-large reflects whether a prediction model satisfies *good calibration*, meaning that the expected value of the risk should be as close as possible to the prevalence $\theta = P(D = 1)$. It follows that the risks should satisfy $\theta - E(risk(Y)) = 0$ when *good calibration* is satisfied. The empirical counterpart of $\theta - E(risk(Y))$ is

$$\hat{A}_{\text{RBP}} := \frac{1}{n}\sum_{i=1}^{n} D_i - \frac{1}{n}\sum_{i=1}^{n} \hat{p}_i = \frac{1}{n}\sum_{i=1}^{n} \hat{\epsilon}_i.$$

If $\hat{A}_{\text{RBP}} = 0$, *good calibration* is satisfied. Visually, *good calibration* implies that the area above the horizontal zero line and the area below the horizontal zero line should be approximately equal. This is because the integral below the RBP curve can be rewritten as

$$\begin{aligned}\hat{A}_{\text{RBP}} &= \frac{1}{n}\sum_{i=1}^{n}(D_i - \hat{p}_i) \\ &= \frac{1}{n}\left\{\sum_{i:D_i=0}(D_i - \hat{p}_i) + \sum_{i:D_i=1}(D_i - \hat{p}_i)\right\} \\ &= \frac{1}{n}\sum_{i:D_i=0}(0 - \hat{p}_i) + \frac{1}{n}\sum_{i:D_i=1}(1 - \hat{p}_i),\end{aligned}$$

where the two sums correspond to the integrals below and above the horizontal zero line, respectively. For example, in Figure 2(a) the integrals below and above the horizontal zero line are $\frac{1}{n}\sum_{i:D_i=0}(-\hat{p}_i) = -0.0611$ and $\frac{1}{n}\sum_{i:D_i=1}(1 - \hat{p}_i) = 0.0611$, respectively.

*Relation to the calibration across deciles.* In a similar way one can show that the RBP curve is related to the calibration across deciles of the predicted risks. Let $Q_q$, with $q = 0.1, 0.2, \ldots, 1$, be the sets of observations whose predicted risks are included in intervals that are bounded by the $(q * 10 - 1)$-th and $(q * 10)$-th deciles of the predicted risks, so that the sets are equally sized. Furthermore, let $\hat{A}^q_{RBP}$, $q = 0.1, 0.2, \ldots, 1$, be the integrals that are obtained by splitting $\hat{A}_{RBP}$ according to deciles of the predicted risks, so that $\hat{A}_{RBP} = \hat{A}^{0.1}_{RBP} + \hat{A}^{0.2}_{RBP} + \cdots + \hat{A}^{1}_{RBP}$. For example, the integral for the first decile is estimated by

$$\hat{A}^{0.1}_{RBP} = \frac{1}{n_{0.1}}\sum_{i\in Q_{0.1}}(D_i - \hat{p}_i) = \frac{1}{n_{0.1}}\sum_{i\in Q_{0.1}} D_i - \frac{1}{n_{0.1}}\sum_{i\in Q_{0.1}} \hat{p}_i,$$

where $n_{0.1} = |Q_{0.1}|$ is the number of observations below the first decile of predicted risks. Analogous to the previous section, this integral can be rewritten as

$$\hat{A}^{0.1}_{RBP} = \frac{1}{n_{0.1}}\sum_{\{i:i\in Q_{0.1}\wedge D_i=0\}}(-\hat{p}_i) + \frac{1}{n_{0.1}}\sum_{\{i:i\in Q_{0.1}\wedge D_i=1\}}(1 - \hat{p}_i),$$

where the first sum corresponds to the integral below the horizontal zero line (using only the observations whose predicted risks are below the first decile). The second sum corresponds to the integral above the horizontal zero line, also using only the observations whose predicted risks are below the first decile. When both integrals are equal in magnitude, they sum up to zero, yielding $\hat{A}^{0.1}_{RBP} = 0$, so that *good calibration* for the first decile is satisfied. A *well calibrated* prediction model in terms of the calibration across deciles is given when each integral $\hat{A}^{0.1}_{RBP}, \hat{A}^{0.2}_{RBP}, \ldots, \hat{A}^{1}_{RBP}$ is close to zero. In Figure 2(b), areas with the same gray tone refer to the predicted risks for a specific decile. In case of a *well calibrated* prediction model, the areas with the same gray tones above and below the horizontal zero line should therefore be approximately equal.

*Relation to the $L_1$ loss.* There is also a relation between the RBP curve and the absolute error

$$MAE = \frac{1}{n}\sum_{i=1}^{n} L_1(D_i, \hat{p}_i) = \frac{1}{n}\sum_{i=1}^{n}|D_i - \hat{p}_i| = \frac{1}{n}\sum_{i=1}^{n}|\epsilon_i|. \quad (2)$$

As seen from the definition of the residuals $\epsilon_i = D_i - \hat{p}_i$, the RBP curve depicts the signed summands of the $L_1$ loss in equation (2). Visually, the MAE reflects the sum of the absolute values of the integrals below and above the RBP curve, that is, the area enclosed by the RBP curve is proportional to the MAE. For example, in Figure 2(a) the area below and above the horizontal line are both 0.0611, yielding $MAE = 0.0611 + 0.0611 = 0.1222$. Consequently, although the exact value of the MAE cannot be read off directly from the RBP curve, inspecting the area enclosed by the RBP curve provides analysts with information on the magnitude of the MAE.

*Relation to the MRD.* The MRD measure is visually obtained by the difference of the gray areas in Figure 2(c). By definition, the gray shaded area above the horizontal line corresponds to the conditional expectation $E_1 := E(risk(Y) \mid D = 1)$ in the diseased group (estimated by $\hat{E}_1 = \frac{1}{n_1}\sum_{i:D_i=1}\hat{p}_i$), whereas the gray shaded area below the horizontal line corresponds to the conditional expectation $E_0 := E(risk(Y) \mid D = 0)$ in the non-diseased group (estimated by $\hat{E}_0 = \frac{1}{n_0}\sum_{i:D_i=0}\hat{p}_i$). It follows that the MRD measure is estimated by $\widehat{\text{MRD}} = \hat{E}_1 - \hat{E}_0$. For example, in Figure 2(c) one obtains $\widehat{\text{MRD}} = \hat{E}_1 - \hat{E}_0 = 0.796 - 0.087 = 0.709$.

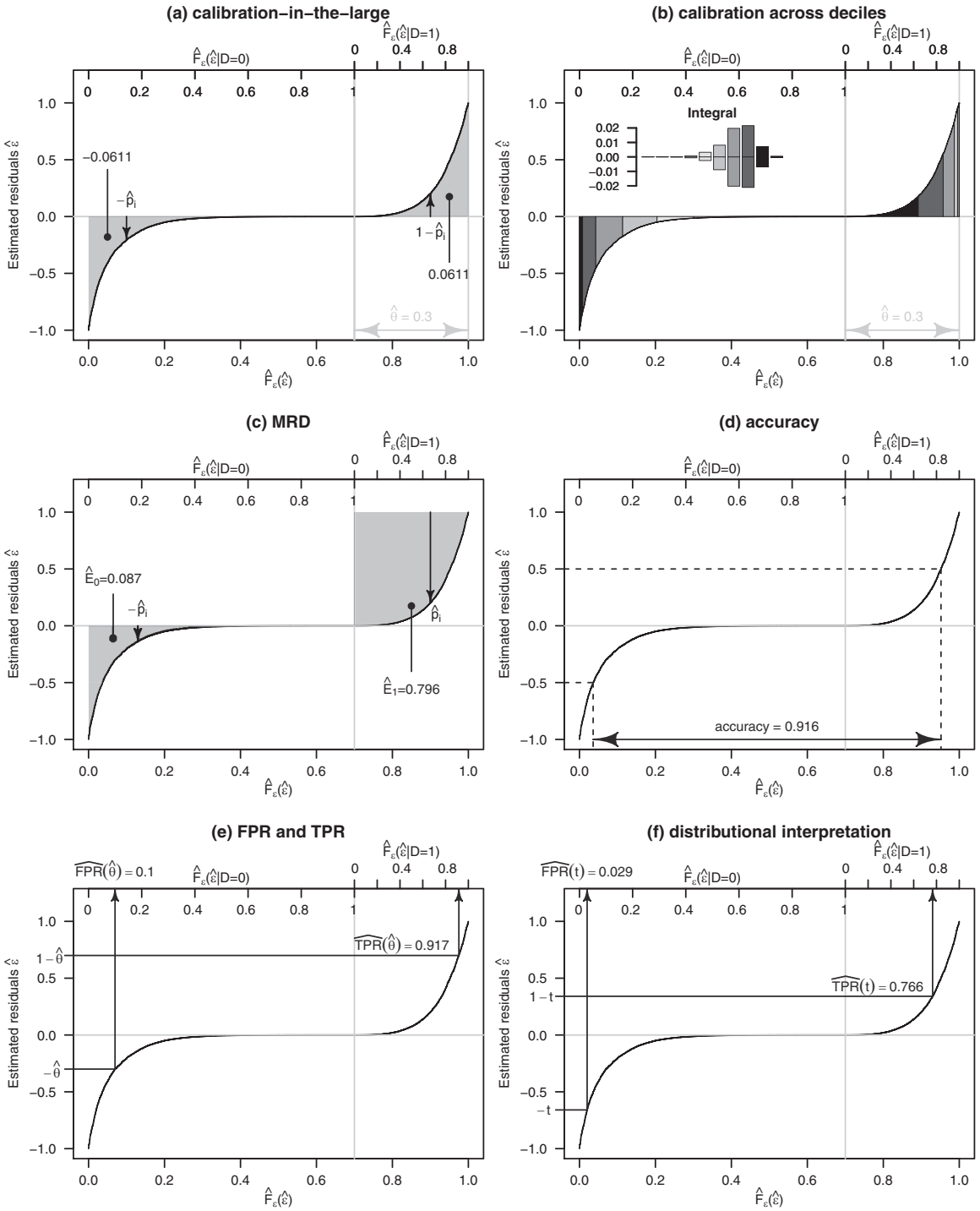



**Figure 2.** Derivation of performance measures from the RBP curve, as described in Section 3. The plots are based on the simulated data and the predictions from the logistic model described in Section 1. The numbers in panels (a) correspond to the integral of the respective regions. The bar plots in the panel (b) facilitate the comparison of integrals with the same gray tones, which should be equal in magnitude. That is, the bar plots should be symmetric about the x-axis when the predicted risks are *well calibrated*.

*Relation to classification accuracy.* The classification accuracy is defined as

$$ACC = \frac{1}{n}\sum_{i=1}^{n} I(\hat{D}_i = D_i),$$

where $\hat{D}_i$ is the predicted outcome calculated from the predicted risks $\hat{p}_i$ using a threshold $t$, so that $\hat{D}_i = I(\hat{p}_i > t)$. Typically $t$ is set to 0.5. For given $t$, the classification accuracy can be visually assessed via the RBP curve by the proportion of points lying above $-t$ and below $1-t$ (Figure 2(d)).

*Relation to the FPR and TPR.* True and false positive rates are defined by

$$\begin{aligned} TPR(\theta) &= P(risk(Y) > \theta \mid D = 1) \\ &= P(1 - risk(Y) < 1 - \theta \mid D = 1) \\ &= F_\epsilon(1 - \theta \mid D = 1), \\ FPR(\theta) &= P(risk(Y) > \theta \mid D = 0) \\ &= P(0 - risk(Y) < 0 - \theta \mid D = 0) \\ &= F_\epsilon(0 - \theta \mid D = 0), \end{aligned}$$

respectively. Estimates of FPR and TPR are hence given by $\widehat{TPR}(\hat{\theta}) = \hat{F}_\epsilon(1-\hat{\theta} \mid D = 1)$ and $\widehat{FPR}(\hat{\theta}) = \hat{F}_\epsilon(0-\hat{\theta} \mid D = 0)$, which correspond to the intersections of the RBP curve with the horizontal lines at $0-\hat{\theta}$ and $1-\hat{\theta}$, respectively (see Figure 2(e)).

*Relation to the total gain.* Bura and Gastwirth (2001) proposed the standardized total gain, which is defined as $\overline{TG} = TPR(\theta) - FPR(\theta)$ and can be estimated by $\widehat{\overline{TG}} = \widehat{TPR}(\hat{\theta}) - \widehat{FPR}(\hat{\theta})$ (see also Gu and Pepe, 2009). As discussed in the previous paragraph, the components $\widehat{TPR}(\hat{\theta})$ and $\widehat{FPR}(\hat{\theta})$ can be directly obtained from the RBP curve. It is therefore possible to "read off" these values from the RBP curve and to estimate the standardized total gain by $\widehat{\overline{TG}} = \widehat{TPR}(\hat{\theta}) - \widehat{FPR}(\hat{\theta})$.

*Distributional information.* As shown in Section 1, the predictiveness curve contains the full distribution of the risk $R(v)$, which is depicted as a function of the quantiles of the marker $Y$. It follows that one can read off the proportion of the population having a disease risk less than or equal to a specific value $t$ (see also Figure 1). Analogous distributional interpretations can be derived from the RBP curve. This is seen when writing $P(risk(Y) \le t)$ in terms of TPR, FPR, and the prevalence $\theta$, whose values can all be obtained from the RBP curve:

$$\begin{aligned} P(risk(Y) \le t) &= 1 - [P(risk(Y) > t \mid D = 1) \cdot P(D = 1) \\ &\quad + P(risk(Y) > t \mid D = 0) \cdot \{1 - P(D = 1)\}] \\ &= 1 - \left\{TPR(t) \cdot \theta + FPR(t) \cdot (1 - \theta)\right\}. \end{aligned}$$

The values of $TPR(t)$ and $FPR(t)$ can be read off at the intersection of the curve and the horizontal lines at $1-t$ and $-t$. In Figure 2(f), for example, we used $t = 0.66$ and obtained $TPR(t) = 0.766$ and $FPR(t) = 0.029$. It follows that $P(risk(Y) \le t) = 0.75$, yielding the same distributional interpretation as in Figure 1, namely that 75% of the population have a risk $\le 66\%$.

*Relation to the ROC curve.* The ROC curve is constructed by plotting all pairs $(FPR(t), TPR(t))$ at each possible decision threshold $t$ (e.g. Zou et al., 1997). As illustrated in Figure 2(e), it is possible to read the estimated values of $FPR(t)$ and $TPR(t)$ off the RBP curve. One can, therefore, calculate the ROC curve from the RBP curve by reading off and plotting the pairs $(FPR(t), TPR(t))$. Generally, the closer the RBP curve to the horizontal line at 0, the smaller the FPR and the higher the TPR values, yielding a steeper ROC curve. From Figure 2(e) and Figure 2(f), for example, we obtained the pairs $(TPR(\hat{\theta}), FPR(\hat{\theta})) = (0.917, 0.1)$ and $(TPR(t), FPR(t)) = (0.766, 0.029)$, so that the resulting ROC curve will pass through these points.

*Relation to the decision curve.* Decision curve analysis has been proposed by Vickers and Elkin (2006) to incorporate the relative harms of false positive and false negative predictions into the evaluation of a prediction model. The decision curve is the plot of the so-called "net benefit" $NB(t)$ versus a specific threshold $t$ of the marker $Y$. Janes and Pepe (2013) showed that the net benefit can be written in terms of $TPR(t)$, $FPR(t)$ and the prevalence $\theta$: $NB(t) = \theta\, TPR(t) - \frac{t}{1-t}(1-\theta)\, FPR(t)$. It is therefore possible to derive an estimate of $NB(t)$ at a given threshold $t$ from the RBP curve. This is done by reading off the components $\widehat{TPR}(t)$ and $\widehat{FPR}(t)$ from the curve (see Figure 2(f)) and by inserting them into the above formula.

Remark 1. *The above considerations are valid as long as $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} D_i$ is a valid estimate of the prevalence $\theta$. This is, for example, the case when the data are an i.i.d. sample of the underlying population. In contrast, if no valid estimate of $\theta$ can be obtained from the data, prevalence-based measures such as the calibration-in-the-large and the total gain cannot be read off the RBP curve in a meaningful way. A typical example in this respect are case-control studies, which are usually based on a pre-defined ratio of cases and controls.*

## 4. Simulations

In this section, we present the results of two simulation studies. In the first study (Section 4.1), we illustrate the importance of calibration checks before assessing the performance of a prediction model using the predictiveness curve. By evaluating a model that massively overfits the data, we demonstrate that calibration checks are not possible without consideration of the outcome variable $D$ and that the predictiveness curve hence cannot visualize calibration problems. In the second study (Section 4.2), we show that it is possible to construct models with almost undistinguishable predictiveness curves although one of the models ignores an influential marker. In contrast to the predictiveness curves, the respective RBP curves visualize both overfitting issues and the inclusion/exclusion of influential markers.

### 4.1. *Overfitting Issues*

An overfitting prediction model fits a specific data set well but poorly predicts future values of $D$. To construct such a model, we generated a data set $\mathcal{D} = \{(\mathbf{x}_i, D_i), i = 1, \ldots, n\}$ with $n = 3000$ observations and 1000 non-informative markers $\mathbf{x}_i^\top = (x_{i,1}, \ldots, x_{i,1000})^\top$. The data were subdivided into

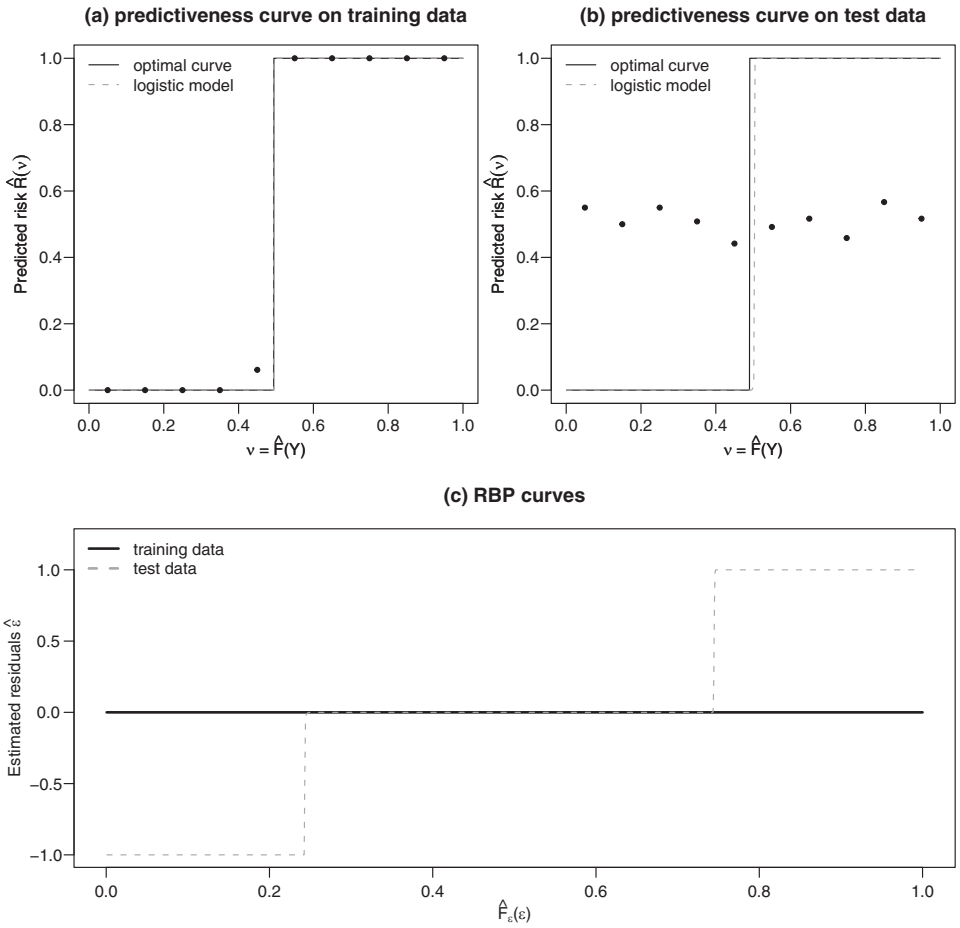



**Figure 3.** Predictiveness curves and RBP curves obtained from the overfitting prediction model considered in Section 4.1. The points correspond to the observed proportions of diseased subjects in the risk deciles.

a training set $\mathcal{L} = \{(\mathbf{x}_i, D_i), i = 1, \ldots, 1800\}$ and a test set $\mathcal{T} = \mathcal{D} \setminus \mathcal{L}$. Based on the training data, we estimated the coefficients of a logistic regression model that included all available predictors. The model was subsequently evaluated on the test data. Panels (a) and (b) in Figure 3 show the predictiveness curves for the predicted risks in the training and test data. It is seen that the shapes of both curves are similar to the respective "optimal" predictiveness curve, suggesting an "optimal" predictive performance of the model. On the other hand, when considering the calibration points in Figure 3(b) (obtained from the test data), it is obvious that the prediction model is not *well calibrated* at all. In particular, Figure 3(a) and (b) show that no valid calibration and performance checks are possible without adding the calibration points to the curve.

Figure 3(c) shows the RBP curves obtained from both the training and the test data. As expected, the RBP curve in the training data is a horizontal line at zero. As it corresponds to a prediction model with extreme overfitting, it is very similar to the "optimal" RBP curve defined in Section 3. Unlike the predictiveness curve obtained from the test data depicted in Figure 3(b), the respective RBP curve looks far from being optimal. Because the RBP curve depends on the real outcome values, it is sensitive to bad calibration and large prediction errors.

In Web Appendix B, we demonstrate that it is even possible to optimize the predictiveness curve of a model with poor performance such that it still looks "optimal" as in Figure 1. This can be achieved by direct maximization of the empirical MRD measure.

### 4.2. *Model Comparisons*

In this section we compare two *well calibrated* prediction models (one model with and the other model without an influential marker) by their predictiveness curves. The aim of our simulation study was to demonstrate that making conclusions about the model performance by solely using the predictiveness curve can be misleading – even if the prediction model is *well calibrated*.

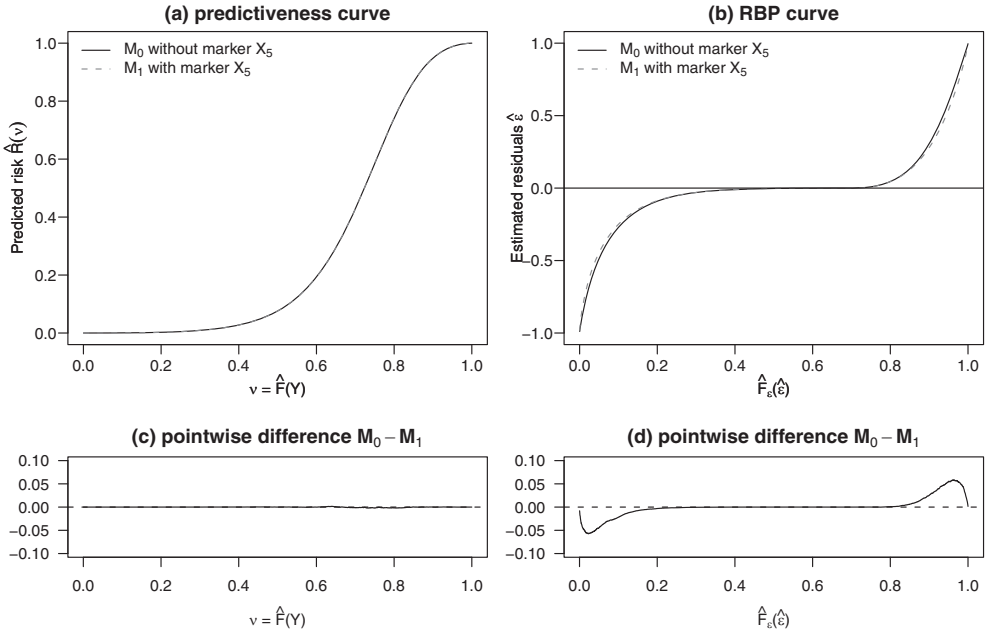



**Figure 4.** Results of the simulation study of Section 4.2. Panels (a) and (c) show the averaged predictiveness curves for models $M_0$ and $M_1$ and the pointwise differences between the two predictiveness curves, respectively. Panels (b) and (d) visualize the averaged RBP curves and their pointwise differences, respectively.

We simulated 1000 data sets with 10,000 observations each and considered five independent markers $X_1, \ldots, X_5 \sim N(0,1)$. The marker combination $\eta = \mathbf{x}^\top \boldsymbol{\beta}$ was defined by the randomly generated coefficients $\boldsymbol{\beta} = (-2.71, -2.21, -2.59, -2.12, -2.06, 1)^\top$, where the first coefficient corresponded to the intercept and the others corresponded to the influential markers. Each of the 1000 data sets was split into training and test sets (containing 60 and 40% of the observations, respectively). In each training set we fitted one model with marker $X_5$ (model $M_1$) and one model without $X_5$ (model $M_0$). By solving the optimization problem from Web Appendix B, we ensured that the model fits were "optimal" with respect to the step function in Figure 1. Details on model fitting and tuning parameter selection are given in Web Appendix C.1.

In the next step we applied the Hosmer–Lemeshow test to both the training and the test data and selected those data sets whose predicted risks were *well calibrated* (according to the Hosmer–Lemeshow test) for both the training and the test data. Although the Hosmer–Lemeshow test has been criticized in many respects (Kramer and Zimmerman 2007), using this test in a simulation study nicely illustrates that model comparisons based on the predictiveness curve are problematic even when automated calibration checks are carried out beforehand. In our simulation study, 114 out of the 1000 data sets passed the Hosmer–Lemeshow test. We used these 114 data sets for the following computations.

Figure 4 shows the averaged predictiveness curves obtained from the 114 test data sets (panel (a)) as well as the respective pointwise differences between the averaged predictiveness curves (panel (c)). Since the curves almost overlap and the pointwise differences scatter around the zero line, the model performance indicated by the predictiveness curves is similar for the two models. It is therefore difficult to tell whether the model with or without $X_5$ performed better, although, according to the data generating process, the former model includes the influential predictor $X_5$.

In contrast to panels (a) and (c), panels (b) and (d) of Figure 4 (which are based on the RBP curve instead of the predictiveness curve) indicate a difference between models $M_0$ and $M_1$. Specifically, panel (d) indicates that model $M_1$, which reflects the true data generating process, has a better model performance than model $M_0$. This result was confirmed by the evaluation of various other performance measures on the test data (see Web Figure 2). Note that the aim of Figure 4 was not to analyze and understand a single data set but rather to work out the systematic differences between the original predictiveness curve and the RBP curve. We also repeated this simulation with smaller sample sizes and obtained similar results (see Web Appendix C.3).

Remark 2. *The differences between RBP curves shown in Figure 4(d) are related but not equivalent to a method proposed by Uno et al. (2011). The authors suggested to compare competing prediction models by evaluating the differences in predicted risk for any given marker combination. Although this approach is similar to what is shown in Figure 4(d), we emphasize that the RBP procedure* sorts *the respective sets of residuals in increasing order before differences are taken. Therefore, the difference between two predictiveness curves at*

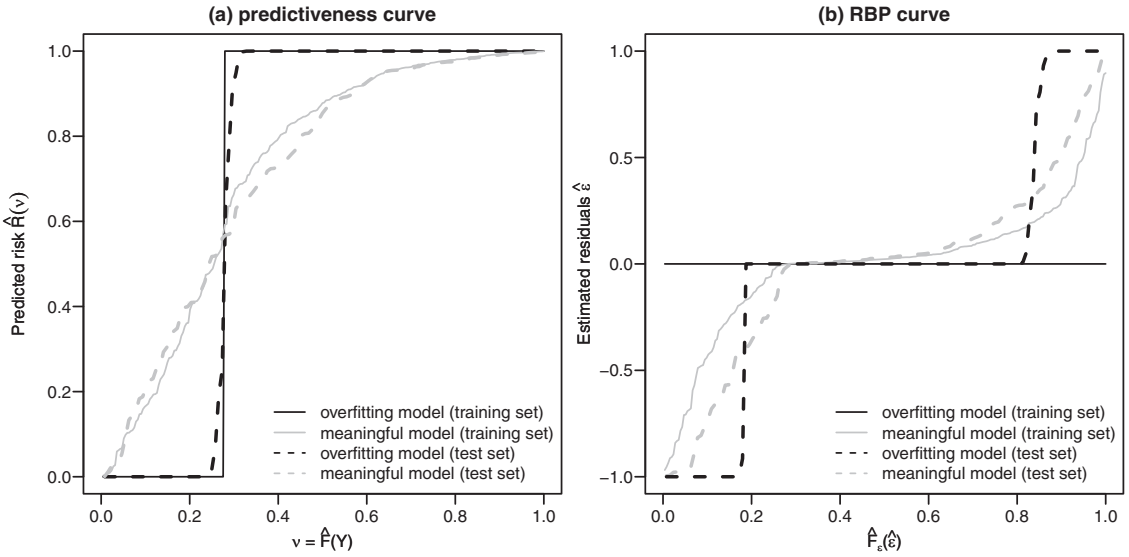



**Figure 5.** Predictiveness curves (panel (a)) and RBP curves (panel (b)) obtained from the training and test data in Section 5.

*a specific value on the abscissa does not necessarily refer to a single observation or a unique covariate combination. In contrast, the method by Uno et al. (2011) is based on the* observation-wise *differences in predicted risks.*

## 5. Evaluation on Real Data

The data set considered in this section was collected by Hatzis et al. (2011). It contains information on patients with newly diagnosed ERBB2-negative breast cancer and is publicly available from the GEO repository website at http://www.ncbi.nlm.nih.gov/geo. Covariate information consists of clinical data and high-throughput molecular data (expression levels of 22,283 genes, see Hatzis et al. (2011) for details). The data are stored in two separate data sets (GEO accession numbers GSE25055 and GSE25065) that, in our analysis, were merged together and randomly split into training set and test set containing 2/3 and 1/3 of the observations, respectively. The response variable considered for our analysis was the residual cancer burden (RCB) class. The levels RCB-0 and RCB-I (referring to no and minimal residual disease, respectively) were coded as $D = 0$, whereas levels RCB-II and RCB-III (referring to moderate and extensive residual disease, respectively) were coded as $D = 1$. Observations with missing values in the clinical data and/or the response (RCB class) were omitted, yielding training and test sets of sizes 254 and 128, respectively.

To construct a meaningful model that combines clinical and omics information, we used an approach that was originally proposed by De Bin, Sauerbrei, and Boulesteix (2014) ("Strategy 4a"). The main idea of this strategy is to first fit a model to the omics data and to use the linear predictor obtained from this model, the so-called *omics score*, as an additional explanatory variable in a final logistic regression model that also contains the clinical predictors age, nodal status, tumor size, grade, estrogen receptor status, and progesterone receptor status. For the derivation of the omics score, we applied a modeling strategy that was able to handle high-dimensional data. More specifically, we used a component-wise gradient boosting algorithm with linear base-learners (Friedman, 2001; Mayr et al., 2014) that automatically selected the most relevant predictors from the data. To avoid overfitting, the number of boosting iterations, which is the main tuning parameter of the algorithm, was optimized using stratified 25-fold bootstrap.

Figure 5 shows the RBP curve and the predictiveness curve for the *meaningful* model, where the omics score was computed using the bootstrapped number of 19 boosting iterations, and for an *overfitting* model, where the omics score was computed using 200 boosting iterations. It can be seen that the RBP curve displays the differences in model performance more clearly than the predictiveness curve, especially in case of the *overfitting* model (black curves). Web Appendix D is intended for an exemplary application of the RBP curve to this real data. In Web Appendix D.1, we illustrate how to derive the MRD, TPR and FPR values from the RBP curves of the *overfitting* model and the *meaningful* model. Web Appendix D.2 explains how the added value can be visualized by the RBP curves. Here we used the *meaningful* model with and without the *omics score* to investigate its added value and explain how the in-sample and extra-sample error can be assessed by subsampling the training data.

## 6. Summary and Discussion

In biomedical research it is essential for researchers and practitioners to use reliable and unbiased measures of prediction accuracy. Equally important is the availability of suitable *graphical* tools for visualizing prediction accuracy. In this respect, the RBP curve is a convenient choice, since it contains the full distribution of the risks and several well-established summary measures can be derived from it. In particular, the

RBP curve visualizes both discrimination and calibration and is therefore a suitable graphical tool for unbiased marker comparisons.

It is important to note that the RBP curve and the predictiveness curve are based on the same assumptions, namely the existence of a marker or marker combination $Y$ (that is not necessarily restricted to the interval $[0, 1]$) and a probability model that is needed to calculate $risk(Y)$. In contrast to strategies for the original predictiveness curve, the RBP curve does not rely on asymptotically valid calibration tests and is therefore insensitive to power issues and convergence problems.

As demonstrated in the preceding sections, the RBP curve serves as an evaluation tool that is applied to a data set *after* the prediction rule $risk(Y)$ has been derived. As a consequence, the RBP curve can be used to either assess the "in-sample error" or to assess the "extra-sample error" of the prediction rule under consideration. In the former case, the same data would be used to derive the formula for the prediction rule $risk(Y)$ and to calculate the quantities involved in the definition of the RBP curve. In the latter case, the RBP curve would be applied to a test data set that is different from the data used to derive $risk(Y)$. The variability introduced by estimation of the prediction rule can, for example, be investigated by resampling methods and by plotting the set of resampled RBP curves (see Web Appendix D.2).

The RBP curve is implemented in the `R` package `RBPcurve`, which is available at http://cran.r-project.org. The package provides a user-friendly interface to generate RBP curves and enables users to visualize the relationships between the RBP curve and the measures discussed in Section 3.

## 7. Supplementary Material

Web Appendices and Web Figures referenced in Sections 1, 3.1, and 4–6 are available with this article at the *Biometrics* website on Wiley Online Library.

## Acknowledgements

Financial support from LMUexcellent and Deutsche Forschungsgemeinschaft (Project SCHM 2966/1-2) is gratefully acknowledged.

## References

Bura, E. and Gastwirth, J. L. (2001). The binary regression quantile plot: Assessing the importance of predictors in binary regression visually. *Biometrical Journal* **43**, 5–21.

Cook, N. R. (2010). Comment: Measures to summarize and compare the predictive capacity of markers. *The International Journal of Biostatistics* **6**, Article 22.

Crowson, C. S., Atkinson, E. J., and Therneau, T. M. (2014). Assessing calibration of prognostic risk Scores. *Statistical Methods in Medical Research* DOI: 10.1177/0962280213497434.

De Bin, R., Sauerbrei, W., and Boulesteix, A.-L. (2014). Investigating the prediction ability of survival models based on both clinical and omics Data. *Statistics in Medicine* **33**, 5310–5329.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics* **29**, 1189–1232.

Gu, W. and Pepe, M. S. (2009). Measures to summarize and compare the predictive capacity of markers. *The International Journal of Biostatistics* **5**, 1–49.

Hatzis, C., Pusztai, L., Valero, V., Booser, D. J., Esserman, L., Lluch, A., et al. (2011). A genomic predictor of response and survival following taxane-anthracycline chemotherapy for invasive breast cancer. *Journal of the American Medical Association* **305**, 1873–1881.

Huang, Y., Pepe, M. S., and Feng, Z. (2007). Evaluating the predictiveness of a continuous marker. *Biometrics* **63**, 1181–1188.

Janes, H. and Pepe, M. (2013). Re: "Clinical usefulness of the Framingham cardiovascular risk profile beyond its statistical performance: The Tehran lipid and glucose study". *American Journal of Epidemiology* **177**, 864–865.

Johnson, E. S., Weinstein, J. R., Thorp, M. L., Platt, R. W., Petrik, A. F., Yang, X., et al. (2010). Predicting the risk of hyperkalemia in patients with chronic kidney disease starting lisinopril. *Pharmacoepidemiology and Drug Safety* **19**, 266–272.

Kramer, A. A. and Zimmerman, J. E. (2007). Assessing the calibration of mortality benchmarks in critical care: The Hosmer-Lemeshow test revisited. *Critical Care Medicine* **35**, 2052–2056.

Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014). The evolution of boosting algorithms. *Methods of Information in Medicine* **53**, 419–27.

Moons, K. G. M., Royston, P., Vergouwe, Y., Grobbee, D. E., and Altman, D. G. (2009). Prognosis and prognostic research: What, why, and how? *British Medical Journal* **338**, 1317–1320.

Pepe, M. S. (2010). Rejoinder to N. Cook's Comment on "Measures to summarize and compare the predictive capacity of markers". *The International Journal of Biostatistics* **6**, 16–18.

Pepe, M. S., Feng, Z., and Gu, J. W. (2008). Comment on "Evaluating the added predictive ability of a new marker" by M. J. Pencina et al. *Statistics in Medicine* **27**, 173–181.

Pepe, M. S., Feng, Z., Huang, Y., Longton, G., Prentice, R., Thompson, I. M., et al. (2008). Integrating the predictiveness of a marker with its performance as a classifier. *American Journal of Epidemiology* **167**, 362–368.

Pepe, M. S., Kerr, K. F., Longton, G., and Wang, Z. (2013). Testing for improvement in prediction model performance. *Statistics in Medicine* **32**, 1467–1482.

Soto, K., Papoila, A. L., Coelho, S., Bennett, M., Ma, Q., Rodrigues, B., et al. (2013). Plasma NGAL for the diagnosis of AKI in patients admitted from the emergency department setting. *Clinical Journal of the American Society of Nephrology* **8**, 2053–2063.

Uno, H., Cai, T., Tian, L., and Wei, L. J. (2011). Graphical procedures for evaluating overall and subject-specific incremental values from new predictors with censored event time data. *Biometrics* **67**, 1389–1396.

Vickers, A. J. and Elkin, E. B. (2006). Decision curve analysis: A novel method for evaluating prediction models. *Medical Decision Making* **26**, 565–574.

Zou, K., Hall, W., and Shapiro, D. (1997). Smooth non-parametric receiver operating characteristic (ROC) curves for continuous diagnostic tests. *Statistics in Medicine* **16**, 2143–2156.

*Received January* 2015. *Revised October* 2015.
*Accepted October* 2015.

# 7. Visualizing the Feature Importance for Black Box Models

Chapter 7 first provides an overview of common model-agnostic interpretability methods in machine learning. Furthermore, it introduces a local feature importance measure from which two novel visual tools are derived, namely the partial importance (PI) and individual conditional importance (ICI) plots. The visual tools visualize how changes in a feature affect the model performance on average, as well as for individual observations. Moreover, another feature importance measure called the Shapley feature importance (SFIMP) measure is introduced, which can be used to compare the feature importance across different models, since it fairly distributes the overall performance of a model among the features.

**Contributing article:**

Casalicchio, G., Molnar, C., and Bischl, B. (2018). Visualizing the Feature Importance for Black Box Models. *arXiv preprint arXiv:1804.06620.* *https://arxiv.org/abs/1804.06620*. To Appear in: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018.*

**Copyright information:**

**Author contributions:**

The whole paper and the accompanying R package were written by Giuseppe Casalicchio. Christoph Molnar and Bernd Bischl added valuable input and suggested several notable modifications. All authors proofread and revised the paper.

**Supplementary material available at:**

R package: `https://github.com/giuseppec/featureImportance`

# Visualizing the Feature Importance for Black Box Models

Giuseppe Casalicchio (✉), Christoph Molnar, and Bernd Bischl

Department of Statistics
Ludwig-Maximilians-University Munich
Ludwigstraße 33, 80539 Munich, Germany
giuseppe.casalicchio@stat.uni-muenchen.de

**Abstract.** In recent years, a large amount of model-agnostic methods to improve the transparency, trustability, and interpretability of machine learning models have been developed. Based on a recent method for model-agnostic global feature importance, we introduce a local feature importance measure for individual observations and propose two visual tools: partial importance (PI) and individual conditional importance (ICI) plots which visualize how changes in a feature affect the model performance on average, as well as for individual observations. Our proposed methods are related to partial dependence (PD) and individual conditional expectation (ICE) plots, but visualize the expected (conditional) feature importance instead of the expected (conditional) prediction. Furthermore, we show that averaging ICI curves across observations yields a PI curve, and integrating the PI curve with respect to the distribution of the considered feature results in the global feature importance. Another contribution of our paper is the Shapley feature importance, which fairly distributes the overall performance of a model among the features according to the marginal contributions and which can be used to compare the feature importance across different models.

**Keywords:** Interpretable Machine Learning · Explainable AI · Feature Importance · Variable Importance · Feature Effect · Partial Dependence.

## 1 Introduction and Related Work

Machine learning (ML) algorithms such as neural networks and support vector machines (SVM) are often considered to produce black box models because they do not provide any direct explanation for their predictions. However, these methods often outperform simple linear models or decision trees in predictive performance as they can model complex relationships in the data. Nevertheless, such simple models are still preferred in areas such as life sciences and social sciences due to their simplicity and interpretability [14]. Many researchers have therefore developed and implemented several model-agnostic interpretability tools, which quantify or visualize feature effects or feature importance [9, 11, 17].

In our context, the terms *feature effect*, feature contribution and feature attribution describe how or to what extent each feature contributes to the *prediction*

of the model, either on a local or a global level. Methods for feature effects include partial dependence (PD) plots [10], individual conditional expectation (ICE) plots [11] and, more recently, SHAP values [15]. These methods visualize or quantify the relationship and contribution of each feature to the prediction of a model without requiring knowledge about the true values of the target variable. A method that measures feature effects based on the Shapley value [19] from coalitional game theory was first presented for classification in [21] and has been extended to regression and global analysis in [22]. Further developments, visualizations, and generalizations were introduced by [15, 16]. Similar work proposing a general notion of a quantity of interest for the characteristic function of the Shapley value and focusing on the joint and marginal contributions of feature sets was introduced by [8].

In biomedical research, for example, measuring the effects of biomedical markers w.r.t. model prediction is as essential as measuring their added value regarding model performance [4]. We use the term *feature importance*[1] to describe how important the feature was for the *predictive performance* of the model, regardless of the shape (e.g., linear or nonlinear relationship) or direction of the feature effect. This implies that measures of feature importance require knowledge of the true values of the target variable. The most prominent approach is the permutation importance introduced by Breiman [3] for random forests. It computes the drop in out-of-bag performance after permuting the values of a feature. A model-agnostic global permutation-based feature importance (PFI) was recently introduced in [9].

*Contributions:* We review model-agnostic global PFI and propose an efficient approximation based on Monte-Carlo integration. We then introduce a local version of the global PFI, which measures the feature importance of individual observations. We provide visualizations for local and global PFI, which illustrate how changes in the considered feature affect model performance. We also relate our new visual tools to PD plots, ICE plots and show that the integral of our PI curve results in the global PFI measure. Furthermore, we propose a permutation-based Shapley feature importance (SFIMP) measure that fairly distributes the model performance among features and allows the comparison of feature importances across different models.

## 2 Preliminaries and Background on Feature Effects

In this section, we introduce the notation and describe methods focusing on feature effects, which we transfer to feature importance in Section 4 and 5.

*General Notation:* Consider a $p-$dimensional feature space $\mathcal{X}_P = (\mathcal{X}_1 \times \ldots \times \mathcal{X}_p)$ with the feature index set $P = \{1, \ldots, p\}$ and a target space $\mathcal{Y}$. Suppose that there is an unknown functional relationship $f$ between $\mathcal{X}_P$ and $\mathcal{Y}$. ML algorithms try to learn this relationship using training data with observations

[1] In the literature, the term feature importance is sometimes also used for methods that only work with model predictions. In our context, however, we would categorize them under feature effects as they do not take into account the model performance.

that have been drawn i.i.d. from an unknown probability distribution $\mathcal{P}$ on the joint space $\mathcal{X}_P \times \mathcal{Y}$. We consider an arbitrary prediction model $\hat{f}$, fitted on some training data to approximate $f$ and analyze it with model-agnostic interpretability methods. Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ be a test data set sampled i.i.d. from $\mathcal{P}$ where $n$ is the number of observations in the test set. We denote the corresponding random variables generated from the feature space by $X = (X_1, \ldots, X_p)$ and the random variable generated from the target space by $Y$. In our notation, the vector $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)})^\top \in \mathcal{X}_P$ refers to the $i$-th observation, which is associated with the target variable $y^{(i)} \in \mathcal{Y}$, and $\mathbf{x}_j = (x_j^{(1)}, \ldots, x_j^{(n)})^\top$ denotes the realizations of the $j$-th feature. We denote the generalization error of a fitted model, which is measured by a loss function $L$ on unseen test data from $\mathcal{P}$, by $GE(\hat{f}, \mathcal{P}) = \mathbb{E}(L(\hat{f}(X), Y))$. It can be estimated using the test data $\mathcal{D}$ by

$$\widehat{GE}(\hat{f}, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^n L(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}). \tag{1}$$

A better estimate for the generalization error of an ML algorithm can be obtained using resampling techniques such as cross-validation or bootstrap [1].

*PD Plots [10]* visualize the marginal relationship between features of interest and the expected prediction of a fitted model on a global level. Consider a subset of feature indices $S \subseteq P$ and its complement $C$. Each observation $\mathbf{x} \in \mathcal{X}_P$ can be partitioned into $\mathbf{x}_S \in \mathcal{X}_S$ and $\mathbf{x}_C \in \mathcal{X}_C$ containing only features from $S$ and $C$, respectively. Let $X_S$ and $X_C$ be the corresponding random variables and let the prediction function using features in $S$, marginalized over features in $C$ be the PD function defined by $f_S(\mathbf{x}_S) = \mathbb{E}_{X_C}(\hat{f}(\mathbf{x}_S, X_C))$. This definition also covers $f_\emptyset(\mathbf{x}_\emptyset)$ and results in a constant, the average prediction over $\mathcal{P}$. We can estimate the PD function using Monte-Carlo integration by averaging over feature values $\mathbf{x}_C^{(i)}$ in order to marginalize out features in $C$:

$$\hat{f}_S(\mathbf{x}_S) = \frac{1}{n} \sum_{i=1}^n \hat{f}_S^{(i)}(\mathbf{x}_S) = \frac{1}{n} \sum_{i=1}^n \hat{f}(\mathbf{x}_S, \mathbf{x}_C^{(i)}). \tag{2}$$

Here, $\hat{f}_S^{(i)}(\mathbf{x}_S) = \hat{f}(\mathbf{x}_S, \mathbf{x}_C^{(i)})$ can be read in two ways: a) the prediction of the $i$-th observation with replaced feature values in $S$ taken from $\mathbf{x}$ or b) the prediction of $\mathbf{x}$ with replaced values in $C$ taken from the $i$-th observation. Plotting the pairs $\{(\mathbf{x}_S^{*(k)}, \hat{f}_S(\mathbf{x}_S^{*(k)}))\}_{k=1}^m$ using (often $m < n$) grid points denoted by $\mathbf{x}_S^{*(1)}, \ldots, \mathbf{x}_S^{*(m)}$ yields a PD curve. Fig. 1 illustrates the PD principle for a simple example.

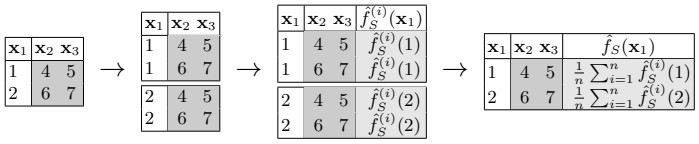



**Fig. 1.** PD plot for an example with $n = 2$, $p = 3$ and $S = \{1\}$ and $C = \{2, 3\}$ (marginal effect of $\mathbf{x}_1$ on $\hat{f}$). We construct a grid using each observed value from $\mathbf{x}_1$, i.e., ${x_1}^{(1)} = 1$ and ${x_1}^{(2)} = 2$, and compute the PD function using these grid points.

*ICE Plots [11]:* The averaging in Eq. (2) of the PD function can obfuscate more complex relationships resulting from feature interactions, i.e. when the partial relationship of one or more observations depends on other features. ICE plots address this problem by visualizing to what extent the prediction of a single observation changes when the value of the considered feature changes. Instead of plotting the pairs $\{(\mathbf{x}_S^{*(k)}, \hat{f}_S(\mathbf{x}_S^{*(k)}))\}_{k=1}^m$, ICE plots visualize the pairs $\{(\mathbf{x}_S^{*(k)}, \hat{f}_S^{(i)}(\mathbf{x}_S^{*(k)}))\}_{k=1}^m$ for each observation indexed by $i \in \{1, \ldots, n\}$.

*Shapley Value:* A coalitional game is defined by a set of players $P$, which can form coalitions $S \subseteq P$. Each coalition $S$ achieves a certain payout. The characteristic function $v : 2^P \to \mathbb{R}$ maps all $2^p$ possible coalitions to their payouts. The Shapley value [19] now fairly assigns a value to each player depending on their contribution in all possible coalitions. This concept was transferred to feature effect estimation in [21]. We could explain the prediction of a single, fixed observation $\mathbf{x}$ by regarding features as players, who form various coalitions (subsets) $S$ to achieve the prediction $\hat{f}(\mathbf{x})$. For each coalition $S$, we are only allowed to access values of features from $S$. A natural definition of the payout is the PD value $f_S(\mathbf{x}_S)$, which we shift so that an empty set of no features is assigned a value of 0 – which is required by the general Shapley value definition:

$$v(\mathbf{x}_S) = \mathbb{E}_{X_C}(\hat{f}(\mathbf{x}_S, X_C)) - \mathbb{E}_X(\hat{f}(X)) = f_S(\mathbf{x}_S) - f_\emptyset(\mathbf{x}_\emptyset). \tag{3}$$

The marginal contribution of feature $j$, joining a coalition $S$, is defined as

$$\Delta_j(\mathbf{x}_S) = v(\mathbf{x}_{S\cup\{j\}}) - v(\mathbf{x}_S) = f_{S\cup\{j\}}(\mathbf{x}_{S\cup\{j\}}) - f_S(\mathbf{x}_S).$$

Let $\Pi$ be the set of all possible permutations over the index set $P$. For a permutation $\pi \in \Pi$, we denote the set of features that are in order *before* feature $j$ as $B_j(\pi)$. For example, for $p = 4$, if we consider feature $j = 4$ and permutation $\pi = \{2, 3, \mathbf{4}, 1\}$, then $B_4(\pi) = \{2, 3\}$. For an observation $\mathbf{x}$ and its feature value for feature $j$, the Shapley value can be estimated by

$$\begin{aligned}\hat{\phi}_j(\mathbf{x}) &= \tfrac{1}{p!} \textstyle\sum_{\pi\in\Pi} \hat{\Delta}_j(\mathbf{x}_{B_j(\pi)}) \\ &= \tfrac{1}{p!} \textstyle\sum_{\pi\in\Pi} \hat{f}_{B_j(\pi)\cup\{j\}}(\mathbf{x}_{B_j(\pi)\cup\{j\}}) - \hat{f}_{B_j(\pi)}(\mathbf{x}_{B_j(\pi)}) \\ &= \tfrac{1}{p!\cdot n} \textstyle\sum_{\pi\in\Pi} \sum_{i=1}^n \hat{f}^{(i)}_{B_j(\pi)\cup\{j\}}(\mathbf{x}_{B_j(\pi)\cup\{j\}}) - \hat{f}^{(i)}_{B_j(\pi)}(\mathbf{x}_{B_j(\pi)}),\end{aligned}$$

where $\hat{f}_{B_j(\pi)}$ and $\hat{f}_{B_j(\pi)\cup\{j\}}$ are estimated by Eq. (2). An efficient approximation based on Monte-Carlo integration using $m$ rather than $p! \cdot n$ summands was proposed by [22]. Consider the following example to illustrate the Shapley value: The features enter a room in a random order specified by the permutation $\pi$. All features in the room participate in the game, i.e., they contribute to the model prediction. The Shapley value $\phi_j$ is the average additional contribution of feature $j$ by joining whatever features already entered the room before.

## 3 Permutation-based Feature Importance

*Background.* The permutation importance for random forests introduced in [3] measures the performance, e.g., the mean squared error (MSE), of each tree

within a random forest using out-of-bag samples. The performance is measured once with and once without permuted values of the feature of interest. The difference between those two performance values is computed for each tree and averaged to yield the feature importance. Permuting the values of a feature breaks the association between the feature and the target variable and results in a large drop in performance if the considered feature is important. A model-agnostic global PFI for features included in $S$ can be defined as

$$PFI_S = \mathbb{E}(L(\hat{f}(\tilde{X}_S, X_C), Y)) - \mathbb{E}(L(\hat{f}(X), Y)) \tag{4}$$

where $\tilde{X}_S$ refers to an independent replication of $X_S$, which is also independent of $X_C$ and $Y$. This implies that $\tilde{X}_S$ is a new (multivariate) random variable, which is distributed as $X_S$, but independent of everything else. This definition is analogous to the permutation-based model reliance introduced by [9] and relates to the definition in [12] where the authors focus on random forests. The larger the value of $PFI_S$, the more substantial the increase in error when we permute feature values in $S$, and the more important we deem the feature set $S$. According to [9], the use of the ratio $PFI_S = \mathbb{E}(L(\hat{f}(\tilde{X}_S, X_C), Y))/\mathbb{E}(L(\hat{f}(X), Y))$ instead of the difference might be more comparable across different models, as it always refers to the relative drop in performance with respect to the standard generalization error. However, using the ratio can result in numerically unstable estimations if the denominator is close or equal to zero. Thus, both definitions have drawbacks that we try to are address in Section 5.

*Estimating and Approximating the PFI.* The first term of Eq. (4) encodes the expected generalization error under perturbation of features in feature set $S$, which can be formulated as:

$$\begin{aligned}\mathbb{E}(L(\hat{f}(\tilde{X}_S, X_C), Y)) &= \mathbb{E}_{(X_C,Y)}(\mathbb{E}_{\tilde{X}_S|(X_C,Y)}(L(\hat{f}(\tilde{X}_S, X_C), Y))) \\ &= \mathbb{E}_{(X_C,Y)}(\mathbb{E}_{\tilde{X}_S}(L(\hat{f}(\tilde{X}_S, X_C), Y))) \\ &= \mathbb{E}_{(X_C,Y)}(\mathbb{E}_{X_S}(L(\hat{f}(X_S, X_C), Y)))\end{aligned}$$

In the derivation above, the first equality follows from the "law of total expectation", the second from the independence of $\tilde{X}_S$ from $(X_C, Y)$, and the third because $\tilde{X}_S$ is distributed as $X_S$. We can plug in an estimator for the inner expected value and denote the estimate of this quantity by

$$\widehat{GE}_C(\hat{f}, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{n}\sum_{k=1}^{n} L(\hat{f}(\mathbf{x}_S^{(k)}, \mathbf{x}_C^{(i)}), y^{(i)}). \tag{5}$$

The index $C$ in $GE_C$ emphasizes that the set of features in $C$ were not replaced with a perturbed random variable and can thus be seen as the model performance using features in $C$ (and ignoring those in $S$). The above estimator is analogous to the V-statistic [18] and may also be replaced by the unbiased U-statistic using $\frac{1}{n}\sum_{i=1}^{n} \frac{1}{n-1}\sum_{k\neq i} L(\hat{f}(\mathbf{x}_S^{(k)}, \mathbf{x}_C^{(i)}), y^{(i)})$ as proposed by [9].[2] The estimator scales

[2] For the sake of simplicity, we consider the V-statistic throughout the article. However, all calculations and approximations based on Eq. (5) still apply – with slight modifications – when using the U-statistic.

with $O(n^2)$ (for a given set $C$, and assuming $\hat{f}$ can be computed in constant time), which can be expensive when $n$ is large. However, we can use a different formulation to motivate an approximation for Eq. (5): Let $\{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{n!}\}$ be the set of all possible permutation vectors over the observation index set $\{1, \ldots, n\}$. As shown by [9], we can replace Eq. (5) by the equivalent formulation

$$\widehat{GE}_{C,\text{perm}}(\hat{f}, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{n!} \sum_{k=1}^{n!} L(\hat{f}(\mathbf{x}_S^{(\tau_k^{(i)})}, \mathbf{x}_C^{(i)}), y^{(i)}).$$

If we approximate $\widehat{GE}_{C,\text{perm}}$ by Monte-Carlo integration using only $m$ randomly selected permutations rather than all $n!$ permutations, we obtain

$$\widehat{GE}_{C,\text{approx}}(\hat{f}, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{k=1}^{m} L(\hat{f}(\mathbf{x}_S^{(\tau_k^{(i)})}, \mathbf{x}_C^{(i)}), y^{(i)}). \tag{6}$$

The approximation refers to permuting features in $S$ repeatedly (i.e., $m$ times) and averaging the resulting model performances.[3] The PFI from Eq. (4) can be estimated using Eq. (5) for the first term and using Eq. (1) for the last term. Including the summands into an iterated sum yields the estimate

$$\widehat{PFI}_S = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{k=1}^{n} \left( L(\hat{f}(\mathbf{x}_S^{(k)}, \mathbf{x}_C^{(i)}), y^{(i)}) - L(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}) \right). \tag{7}$$

If we use Eq. (6) rather than Eq. (5), we obtain the approximation

$$\widehat{PFI}_{S,\text{approx}} = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{k=1}^{m} \left( L(\hat{f}(\mathbf{x}_S^{(\tau_k^{(i)})}, \mathbf{x}_C^{(i)}), y^{(i)}) - L(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}) \right). \tag{8}$$

Eq. (8) is identical to the permutation importance of random forests formalized in [12] if we consider $m$ as the number of trees, replace $n$ with the number of out-of-bag samples per tree and replace the model $\hat{f}$ with the individual trees fitted within a random forest, i.e., $\hat{f}_k$.

## 4 Visualizing Global and Local Feature Importance

Consider the summands in Eq. (7) and denote them by

$$\Delta L^{(i)}(\mathbf{x}_S) = L(\hat{f}(\mathbf{x}_S, \mathbf{x}_C^{(i)}), y^{(i)}) - L(\hat{f}(\mathbf{x}^{(i)}), y^{(i)}).$$

This quantity refers to the change in performance between the $i$-th observation with and without replaced feature values $\mathbf{x}_S$. Inspired by ICE plots, we introduce *individual conditional importance* (ICI) plots which visualize the pairs $\{(\mathbf{x}_S^{(k)}, \Delta L^{(i)}(\mathbf{x}_S^{(k)}))\}_{k=1}^{n}$ for all observations $i = 1, \ldots, n$. We define the local feature importance of the $i$-th observation (regarding features in $S$) as the integral of

[3] By the same logic, we could also directly approximate Eq. (5) by summing over $m$ randomly selected feature values for features in $S$ instead of using all of them. We here opted for Eq. (6), due to the in our opinion interesting relation to the random forest permutation importance explained at the end of this section.

the corresponding ICI curve with respect to the distribution of $X_S$. It is estimated by $\widehat{PFI}_S^{(i)} = \frac{1}{n}\sum_{k=1}^{n} \Delta L^{(i)}(\mathbf{x}_S^{(k)})$ and can be interpreted as the expected change in performance of the $i$-th observation after marginalizing its features in $S$. It also refers to the contribution of the $i$-th observation to the global PFI (see later in Eq. (9)). To the best of our knowledge, a similar definition for local feature importance only exists in the context of random forests, e.g., in [7].

Analogous to the PD function from Eq. (2), we introduce the *partial importance* (PI) function as the expected change in performance at a specific value $\mathbf{x}_S$, which can be estimated by $\widehat{PI}_S(\mathbf{x}_S) = \frac{1}{n}\sum_{i=1}^{n} \Delta L^{(i)}(\mathbf{x}_S)$. Consequently, a PI plot visualizes the pairs $\{(\mathbf{x}_S^{(k)}, \widehat{PI}_S(\mathbf{x}_S^{(k)}))\}_{k=1}^{n}$ and refers to the pointwise average of all ICI curves across all observations at fixed grid points $\mathbf{x}_S$.

Fig. 2 illustrates the computation of ICI and PI curves for the first feature. It also shows the $n$ grid points for which $\Delta L^{(i)}(\mathbf{x}_S^{(i)}) = 0 \; \forall i$. We can omit these points by plotting the pairs $\{(\mathbf{x}_S^{(k)}, \Delta L^{(i)}(\mathbf{x}_S^{(k)}))\}_{k \in \{1,\ldots,n\} \setminus \{i\}}$ to visualize the unbiased estimation of the feature importance proposed by [9]. Visualizing the ICI curves for the approximation in Eq. (8) implies that some grid points are randomly skipped because the feature values used as grid points are implicitly determined by the randomly selected permutations in Eq. (8). The ICI curves, the PI curve, and the global PFI are related: Averaging all ICI curves pointwise yields a PI curve. Integrating the PI curve (as well as averaging the integral of all ICI curves) using Monte-Carlo integration over all points $\{\mathbf{x}_S^{(k)}\}_{k=1}^{n}$ yields an equivalent estimate of the global PFI from Eq. (7):

$$\widehat{PFI}_S = \frac{1}{n}\sum_{i=1}^{n} \widehat{PFI}_S^{(i)} = \frac{1}{n}\sum_{k=1}^{n} \widehat{PI}_S(\mathbf{x}_S^{(k)}). \tag{9}$$

We propose to additionally inspect the PI and ICI curves instead of focusing on a single PFI value. PI curves enable the user to identify regions in which the feature importance is higher or lower than its global PFI. ICI curves additionally enable the user to identify (suspicious) observations that impact the global PFI strongly and can reveal heterogeneity in the feature importance among the observations, which remain hidden in the PI plots (see also Section 6).

Algorithm 1 describes a procedure for obtaining PI and PD plots, which also allows to return ICI and ICE plots by visualizing $\{(\mathbf{x}_S^{*(k)}, \Delta L^{(i)}(\mathbf{x}_S^{*(k)}))\}_{k=1}^{m}$ and $\{(\mathbf{x}_S^{*(k)}, \hat{f}_S^{(i)}(\mathbf{x}_S^{*(k)}))\}_{k=1}^{m}$ for all observations $i$. Similar to PD and ICE plots, we can use all $k = 1, \ldots, n$ or a random sample (of size $m < n$) of feature values from $S$ as grid points for PI and ICI plots.

## 5 Shapley Feature Importance

In this section, we introduce the *S*hapley *F*eature *IMP*ortance (SFIMP) measure, which allows to easily visualize and interpret the contribution of each feature to the model performance. Our goal is to fairly distribute the performance difference among the individual features between the scenario when all features are used and when all features are ignored, which is illustrated in Fig. 3.

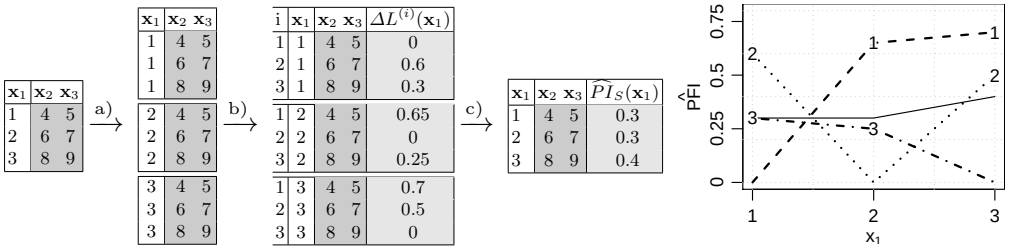



**Fig. 2.** The tables on the left side illustrate the required steps to create ICI curves and PI plots as described in Algorithm 1. The right plot visualizes the ICI curves of individual observations for $i = 1, 2, 3$ (dotted and dashed lines) and the PI curve (solid line) which is the average of ICI curves at each point of the abscissa. All points belonging to the same observation are connected by a line to produce the ICE curves.

**Algorithm 1:** PD plot and PI plot

1. Choose $m$ grid points $\mathbf{x}_S^{*(1)}, \ldots, \mathbf{x}_S^{*(m)}$.
2. Repeat the following steps for the $k$-th grid point:
   a) Modify the data by replacing all observed values in $\mathbf{x}_S$ with the constant values from the $k$-th grid point $\mathbf{x}_S^{*(k)}$.
   b) Use the modified data from a), the prediction function $\hat{f}$ and the loss functic $L$ and calculate for all individual observations:
      i) $\hat{f}_S^{(i)}(\mathbf{x}_S^{*(k)}) = \hat{f}(\mathbf{x}_S^{*(k)}, \mathbf{x}_C^{(i)})$
      ii) $\Delta L^{(i)}(\mathbf{x}_S^{*(k)}) = L(\hat{f}(\mathbf{x}_S^{*(k)}, \mathbf{x}_C^{(i)}), y^{(i)}) - L(\hat{f}(\mathbf{x}^{(i)}), y^{(i)})$
   c) Aggregate the individual values:
      i) $\hat{f}_S(\mathbf{x}_S^{*(k)}) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}_S^{(i)}(\mathbf{x}_S^{*(k)})$
      ii) $\widehat{PI}_S(\mathbf{x}_S^{*(k)}) = \frac{1}{n} \sum_{i=1}^{n} \Delta L^{(i)}(\mathbf{x}_S^{(k)})$
3. Plot the pairs $\{(\mathbf{x}_S^{*(k)}, \hat{f}_S(\mathbf{x}_S^{*(k)}))\}_{k=1}^{m}$ and $\{(\mathbf{x}_S^{*(k)}, \widehat{PI}_S(\mathbf{x}_S^{*(k)})\}_{k=1}^{m}$.

The Shapley value was used in [6] for a fair attribution of the difference in model performance. However, the authors focused on feature selection which requires refitting the model by leaving out or including features. This can lead to different results of the learning algorithm since different relationships can be learned due to the absence of features. This is reasonable in the context of feature selection. However, as we measure the feature importance of an already fitted model, we prefer marginalizing over features rather than omitting them completely. Inspired by Eq. (3), we define the characteristic function of the coalition of features in $S \subseteq P$ based on Eq. (5) as:

$$v_{GE}(S) = \widehat{GE}_S(\hat{f}, \mathcal{D}) - \widehat{GE}_\emptyset(\hat{f}, \mathcal{D}). \tag{10}$$

The characteristic function measures the change in performance between using features in $S$ (i.e., ignoring features in its complement $C$ by marginalizing over them) and ignoring all features. This is similar to Eq. (7) which, in contrast, measures the change in performance between ignoring features in $S$ and using all features. Since the error $\widehat{GE}_\emptyset(\hat{f}, \mathcal{D})$ (no features are considered, i.e., all

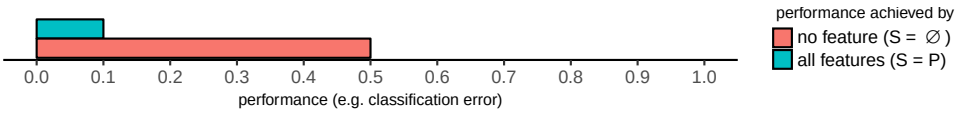



**Fig. 3.** Illustration of the difference in model performance that we want to fairly distribute among the features. The model performance (e.g., classification error) is 0.1 when using all features (green bar) and 0.5 when ignoring all features (red bar). Our goal is to fairly distribute the resulting performance difference of 0.4 among all involved features based on their marginal contribution.

features are marginalized out) is usually greater than $\widehat{GE}_S(\hat{f}, \mathcal{D})$, $v_{GE}(S)$ will have negative values.[4] The marginal contribution of a feature $j$ to a coalition of features in $S$ is given by

$$\Delta_j(S) = v_{GE}(S \cup \{j\}) - v_{GE}(S) = \widehat{GE}_{S\cup\{j\}}(\hat{f}, \mathcal{D}) - \widehat{GE}_S(\hat{f}, \mathcal{D}).$$

If we consider a permuted order $\pi \in \Pi$ of the features, where $B_j(\pi)$ is the set of features occurring before feature $j$, we obtain the Shapley value estimation

$$\begin{aligned} \hat{\phi}_j(v_{GE}) &= \tfrac{1}{p!} \textstyle\sum_{\pi\in\Pi} \Delta_j(B_j(\pi)) \\ &= \tfrac{1}{p!} \textstyle\sum_{\pi\in\Pi} \widehat{GE}_{B_j(\pi)\cup\{j\}}(\hat{f}, \mathcal{D}) - \widehat{GE}_{B_j(\pi)}(\hat{f}, \mathcal{D}), \end{aligned} \tag{11}$$

which refers to the SFIMP measure of feature $j$. Computing Eq. (11) is computationally expensive when the number of features $p$ is large, even if we use the approximation of the model performance from Eq. (6). We therefore suggest an efficient procedure in Algorithm 2. The Shapley value satisfies the following four desirable properties as already worked out in [6]:

1. Efficiency: $\sum_{j=1}^{p} \phi_j = v_{GE}(P)$. All SFIMP values add up to $v_{GE}(P)$, i.e., the difference in performance between the scenario when all features are used and when all features are ignored. This allows us to calculate the proportion of explained importance for each feature $j$ using $\frac{\phi_j}{\sum_{j=1}^{p} \phi_j}$.
2. Symmetry: If $v_{GE}(S \cup \{j\}) = v_{GE}(S \cup \{k\})$ for all $S \subseteq \{1, \ldots, p\} \setminus \{j, k\}$, then $\phi_j = \phi_k$. Two features $j$ and $k$ have the same SFIMP values if their marginal contribution to all possible coalitions is the same.
3. Dummy property: If $v_{GE}(S \cup \{j\}) = v_{GE}(S)$ for all $S \subseteq P$, then $\phi_j = 0$. The SFIMP value of a feature $j$ is zero if its marginal contribution does not change no matter to which coalition $S$ the feature is added.
4. Additivity: $\phi_j(v_{GE} + w_{GE}) = \phi_j(v_{GE}) + \phi_j(w_{GE})$. The SFIMP value resulting from a single game with two combined performance measures $\phi_j(v_{GE} + w_{GE})$ is the same as the sum of the two SFIMP values resulting from two separate games with corresponding characteristic functions, i.e., $\phi_j(v_{GE}) + \phi_j(w_{GE})$. Linearity: $\phi_j(c \cdot v_{GE}) = c \cdot \phi_j(v_{GE})$. Any multiplication of the performance measure with a constant $c$ does not affect the feature ranking.

[4] We prefer the definition in Eq. (10) as it directly shows the relation to Eq. (3), however, we could also swap the sign as discussed at the end of this section.

**Algorithm 2:** Approximation of SFIMP values: Contribution of $j$-th feature towards the model performance.

**Input:** $m_{\text{feat}}$, $m_{\text{obs}}$, $\hat{f}$, $L$, $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
**forall** $k \in \{1, \ldots, m_{\text{feat}}\}$ **do**
  choose a random permutation of the feature indices $\pi \in \Pi$.
  set $S = B_j(\pi)$ containing features that won't be permuted.
  set $\widehat{GE}_{S,\text{perm}} = 0$ and $\widehat{GE}_{S\cup\{j\},\text{perm}} = 0$.
  **forall** $l \in \{1, \ldots, m_{\text{obs}}\}$ **do**
    choose a random permutation of observation indices $\boldsymbol{\tau} \in \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{n!}\}$.
    measure performance by permuting features w.r.t. $\boldsymbol{\tau} = (\tau^{(1)}, \ldots, \tau^{(n)})$:
    $\widehat{GE}_{S,\text{perm}} = \widehat{GE}_{S,\text{perm}} + \frac{1}{n}\sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_S^{(i)}, \mathbf{x}_C^{(\tau^{(i)})}), y^{(i)}))$
    $\widehat{GE}_{S\cup\{j\},\text{perm}} = \widehat{GE}_{S\cup\{j\},\text{perm}} + \frac{1}{n}\sum_{i=1}^{n} L(\hat{f}(\mathbf{x}_{S\cup\{j\}}^{(i)}, \mathbf{x}_{C\setminus\{j\}}^{(\tau^{(i)})}), y^{(i)}))$
  compute marginal contribution for feature $j$ in iteration $k$:
  $\Delta_j^{(k)}(S) = \frac{1}{m_{\text{obs}}} \cdot (\widehat{GE}_{S\cup\{j\},\text{perm}} - \widehat{GE}_{S,\text{perm}})$
**return** $\hat{\phi}_j = \frac{1}{m_{\text{feat}}}\sum_{k=1}^{m_{\text{feat}}} \Delta_j^{(k)}(S)$

The properties above imply that fairly distributing the drop in performance using $v_{PFI}(S) = \widehat{PFI}_S = \widehat{GE}_C(\hat{f}, \mathcal{D}) - \widehat{GE}_P(\hat{f}, \mathcal{D})$ results in the same Shapley values (except for the sign) and is equivalent to using $-v_{GE}(P)$. The SFIMP measure can thus be seen as an extension of the PFI measure in the sense that it additionally fairly distributes the importance values among features. The PFI measure ignores features in $S$ by permuting or marginalizing over them, which destroys any correlation and interaction of features in $C$ with features in $S$. Consequently, the PFI of a feature also includes the importance of any interaction with that feature and features in $C$ and therefore an interaction will be fully attributed to all involved features. The SFIMP measure solves this issue as it considers the marginal contribution of a feature and equally distributes the importance of interactions among the interacting features. This allows comparing feature importances across different models.

## 6 Simulations and Application

For full reproducibility, all our proposed methods are available in the `R` package `featureImportance`[5]. The repository also contains the `R` code, which is partly based on `batchtools` [13], for the application and simulation in this section.

### 6.1 Simulations

*PI and ICI Plots.* Consider the following data-generating model:

$$Y = X_1 + X_2 + 10X_1 \cdot \mathbb{1}_{X_3=0} + 10X_2 \cdot \mathbb{1}_{X_3=1} + \epsilon,$$

[5] https://github.com/giuseppec/featureImportance.

$$X_1, X_2 \overset{i.i.d}{\sim} \mathcal{N}(0,1), X_3 \sim \mathcal{B}(1,0.5), \epsilon \sim \mathcal{N}(0,0.5).$$

We simulate a training data set with 10000 observations, train a random forest and compute the global PFI on 100 test sets of size $n = 100$ sampled from the same distribution. We demonstrate that, by merely inspecting the global PFI, the features $X_1$ and $X_2$ would be considered equally important. However, due to the interactions, it is clear that feature $X_1$ should be considered more important than $X_2$ when $X_3 = 0$ and vice-versa when $X_3 = 1$.

According to Eq. (9), averaging the local feature importances (i.e., the integral of all ICI curves) results in the global PFI. Having at hand the local feature importance of each observation allows calculating the PFI conditional on other features. This does not require additional time-consuming calculations, as we only have to average the already computed local feature importances according to the condition considered in the conditional PFI. The relevance of conditional feature importance in the case of random forests with correlated features was discussed in [20]. In Fig. 4, we illustrate the usefulness of a model-agnostic conditional PFI in case of interactions by showing the PI curves of $X_1$ and $X_2$ conditional on the binary feature $X_3$. The integral of these conditional PI curves refers to the PFI conditional on $X_3$. Its value differs depending on the two groups introduced by feature $X_3$, which suggests that there is an interaction between the features $X_1$ and $X_3$ as well as $X_2$ and $X_3$.

Table 1 shows that feature $X_1$ and $X_2$ are almost equally important if we consider the unconditional global PFI. However, a different ranking of features is obtained when we compute the PFI conditional on $X_3$. Thus, inspecting PI and ICI curves conditional on other feature values may help in detecting interactions.

**Table 1.** The mean and the standard deviation (numbers in brackets) of the PFI values estimated using the 100 simulated test data sets.

| | $X_1$ | $X_2$ |
|---|---|---|
| global PFI | 77.976 (14.15) | 76.764 (13.89) |
| PFI for $X_3 = 0$ | 152.49 (26.06) | 1.428 (1.32) |
| PFI for $X_3 = 1$ | 1.261 (1.03) | 151.489 (24.69) |

*Shapley Feature Importance.* We illustrate how the SFIMP measure can be used to compare the feature importance across different models and present the results of a small simulation study to compare the SFIMP measure introduced in Section 5 with the difference-based and the ratio-based PFI discussed in Section 3. Consider the following data-generating linear model with a simple interaction:

$$Y = X_1 + X_2 + X_3 + X_1 \cdot X_2 + \epsilon, \qquad X_1, X_2, X_3 \overset{i.i.d}{\sim} \mathcal{N}(0,1), \epsilon \sim \mathcal{N}(0,0.5).$$

All three features and the interaction of $X_1$ and $X_2$ have the same linear effect on the target $Y$. We simulate training data with 10000 observations and train four

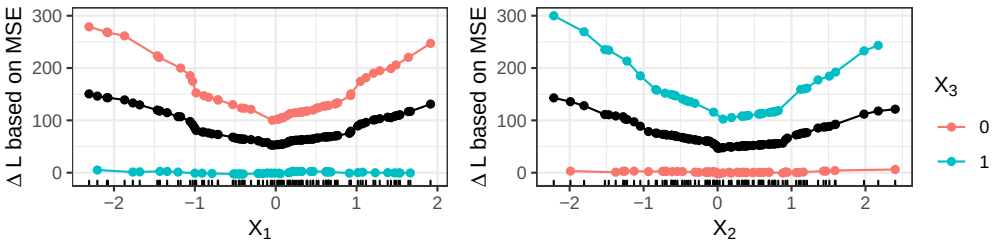



**Fig. 4.** PI curves of $X_1$ and $X_2$ calculated using all observations (black line) and conditional on $X_3 = 0$ (red line) and $X_3 = 1$ (green line). The points plotted on the lines refer to the observed feature values that were used as grid points to produce the corresponding PI curves as described in Algorithm 1.

learning algorithms using the `mlr` R package [2] in their defaults: An SVM with Gaussian kernel (`ksvm`), a random forest (`randomForest`), a simple linear model (`lm`) and another one that considers 2-way interaction effects (`rsm`). We use a test set with $n = 100$ observations sampled from the same distribution and compute the SFIMP values according to Eq. (11). Panel (a) of Fig. 5 displays how the SFIMP measure distributes the total explainable performance among all features and shows the proportion of explained importance for each feature. We repeat the experiment 500 times on different test sets of equal size and additionally compute the difference-based and ratio-based PFI. The results are shown in panel (b) of Fig. 5. For the linear model without interaction effects, the calculated importance of all three features is equal (median ratio of 1). For all other models, we obtained a higher importance for the interacting features, indicating that these models were able to grasp the interaction effect. However, as permuting a feature destroys any interaction with that feature, the PFI values of a feature will also include the importance of any interaction with that feature. Thus, the importance of the interaction between $X_1$ and $X_2$ is contained in the PFI value for feature $X_1$ as well as in the PFI value for feature $X_2$. This will overestimate the importance of $X_1$ and $X_2$ with respect to $X_3$ since $X_1$ and $X_2$ share the same interaction. In panel (b), we thus show the ratio of the importance values with respect to $X_3$. The results suggest that the difference-based PFI considers $X_1$ and $X_2$ twice as important as $X_3$ as the median ratio is around 2. In contrast, the median ratio of SFIMP is around 1.5 as the importance of the interaction is fairly distributed among $X_1$ and $X_2$.

### 6.2 Application on Real Data

We demonstrate our graphical tools on the Boston housing data, which is publicly available on OpenML [23] with data set ID 531. The data set contains 13 features that may affect the median home price of 506 metropolitan areas of Boston. We used the `OpenML` R package [5] and created the OpenML task with ID 167147 containing a holdout split ($\frac{2}{3}$ vs. $\frac{1}{3}$) for training a random forest and producing the PI and ICI plots on the test set.

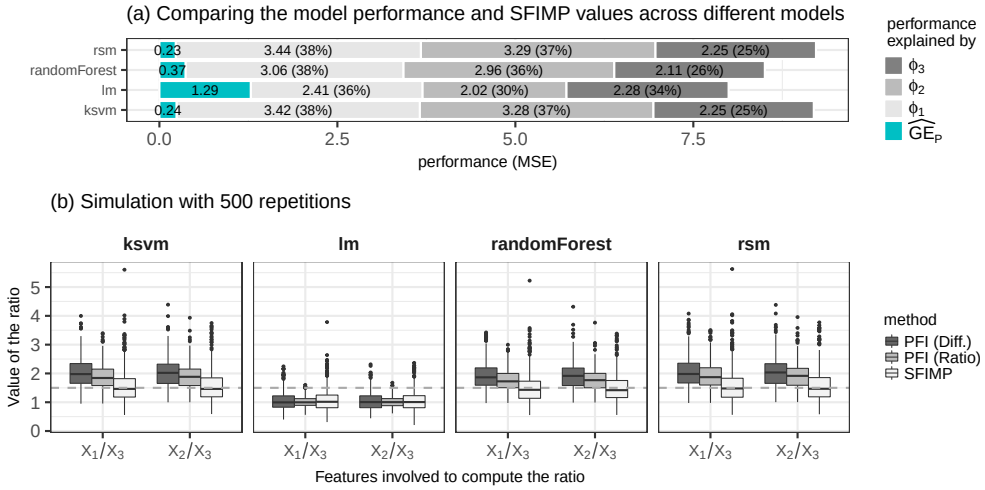



**Fig. 5.** Panel (a) shows the results of a single run, consisting of sampling test data and computing the importance on the previously fitted models. The first numbers on the left refer to the model performance (MSE) using all features. The other numbers are the SFIMP values which sum up to the total explainable performance $v_{GE}(P)$ from Eq. (10). The percentages refer to the proportion of explained importance. Panel (b) shows the results of 500 repetitions of the experiment. The plots display the distribution of ratios of the importance values for $X_1$ and $X_2$ with respect to $X_3$ computed by SFIMP, by the difference-based PFI, and by the ratio-based PFI.

Row (1) of Table 2 shows the global PFI values of all features. They are estimated using Eq. (7) by taking into account all $166 \cdot 166$ points of the test data. Fig. 6 shows the corresponding PI and ICI curves for the two most important features (LSTAT and RM). They visualize which regions of each feature and which observations have a high impact on the computed PFI values on a global and local level, which follows from the relation in Eq. (9).

PI plots visualize the expected change in performance at each position of the abscissa. An expected change close to zero across the whole range of the feature values suggests an unimportant feature. The PI plot of LSTAT in Fig. 6 suggests that the feature is more important if LSTAT $< 10$. For illustration purposes, we omit all observations for which LSTAT $\geq 10$ and recompute the conditional PFI values, which are displayed in Row (2) of Table 2. The resulting conditional PFI values are smaller, i.e., excluding observations for which LSTAT $\geq 10$ makes the LSTAT feature less important. Note that omitting observations change the empirical distribution of the features and thus also affects the importance of other features when the PFI values are recomputed.

ICI curves additionally reveal the most (and the least) influential observations for the feature importance by considering their integral (see highlighted lines in Fig. 6). We can, for example, omit observations with a negative ICI curve integral. In our test set, we observe 18 of 166 ICI curves with a negative integral

for the LSTAT feature. These observations have a negative impact on the global PFI according to the relation in Eq. (9). We omit them and recompute the PFI values. The results are listed in row (3) of Table 2 and show an increased PFI value for LSTAT.

**Table 2.** PFI values calculated for a random forest trained on the Boston housing training set and using the MSE on the test data. The PFI values in row (1) are based on all observations from the test set, in row (2) on a subset where LSTAT $< 10$ and in row (3) after removing observations having a negative ICI integral.

| | LSTAT | RM | NOX | DIS | CRIM | PTRATIO | AGE | INDUS | TAX | RAD | B | ZN | CHAS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1) | 32.0 | 15.6 | 3.9 | 2.7 | 2.6 | 2.2 | 1.2 | 1.0 | 1.0 | 0.8 | 0.8 | 0.1 | 0.1 |
| (2) | 10.4 | 29.6 | 1.5 | 3.3 | 0.8 | 2.3 | 0.8 | 0.5 | 1.2 | 1.1 | 0.6 | 0.2 | 0.2 |
| (3) | 35.3 | 17.0 | 4.3 | 2.4 | 2.5 | 2.5 | 1.1 | 1.2 | 0.8 | 0.9 | 0.8 | 0.1 | 0.1 |

## 7 Conclusion and Future Work

It is essential for practitioners to peek inside black box models to get a better understanding of how features contribute to model predictions or how they affect the model performance. Model-agnostic visualization methods can simplify this task tremendously. Regarding the feature importance, the PI and ICI curves are a convenient choice for visualizing how features affect model performance. We demonstrated how to disaggregate the global PFI into its individual local PFI components, which enabled us to visualize the feature importance on a local and global level. It also allows practitioners to analyze and compare the feature importance across different groups of observations in the data, e.g., by subsetting the data according to other feature values and computing a conditional feature importance similar to [20] on the subsetted data which may reveal interactions. Another interesting aspect, which we leave for future work, is aggregating the local feature importances of individual observations (i.e., the integral of ICI curves) across different features to obtain a measure for the importance of individual observations. This could be used to find clusters of observations in the data that were important for the model performance similar to [15], but based on feature importance rather than feature effects. Furthermore, it is also possible to disaggregate the Shapley feature importance introduced in Section 5 and produce plots similar to Shapley dependency plots that were recently introduced in [15], but we leave this for future work. Our proposed methods serve as an evaluation tool that is applied to a data set *after* a model has been fitted. As a consequence, our methods can be used to either assess the feature importance based on the "in-sample performance" or based on the "out-of-sample performance" of a fitted model. In the former case, the same data could be used to fit the model and to calculate the quantities involved in the definition of our methods. We focused on the latter case with independent test data. However, we could also investigate

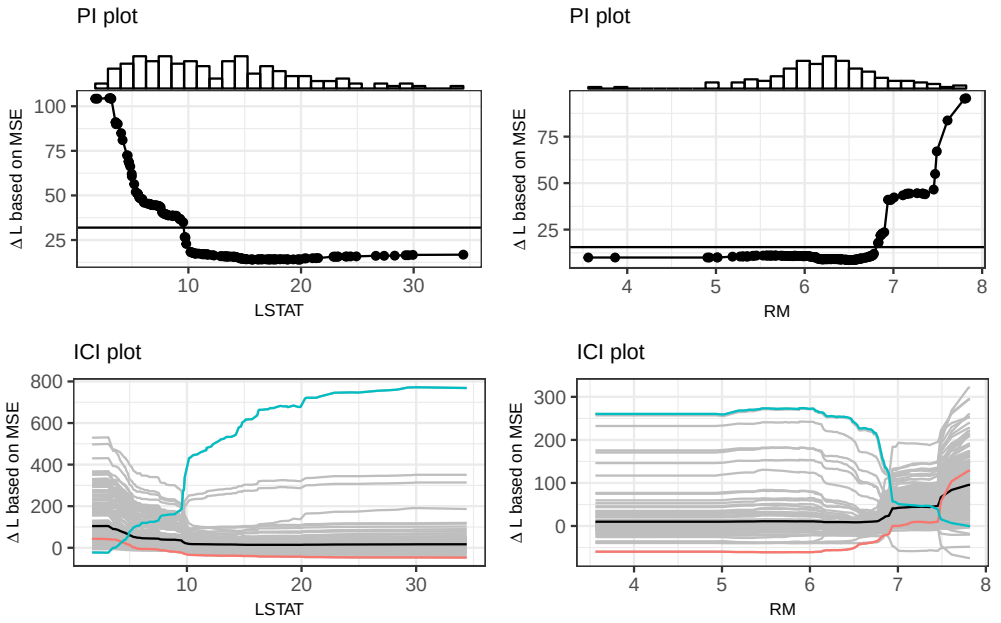



**Fig. 6.** PI and ICI plots for a random forest and the two most important features of the Boston housing data (LSTAT and RM). The horizontal lines in the PI plots represent the value of the global PFI (i.e., the integral of the PI curve). Marginal distribution histograms for features are added to the PI margins. The ICI curve with the largest integral is highlighted in green and the curve with the smallest integral in red.

the variability introduced by the estimation of the model itself via resampling and plot or aggregate the resulting set of quantities.

## Acknowledgments

This work is funded by the Bavarian State Ministry of Education, Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B).

## References

[1] Bischl, B., Mersmann, O., Trautmann, H., Weihs, C.: Resampling methods for meta-model validation with recommendations for evolutionary computation. Evol. Comput. **20**(2), 249–275 (2012)
[2] Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine learning in R. J. Mach. Learn. Res. **17**(170), 1–5 (2016)
[3] Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
[4] Casalicchio, G., Bischl, B., Boulesteix, A.L., Schmid, M.: The residual-based predictiveness curve: A visual tool to assess the performance of prediction models. Biometrics **72**(2), 392–401 (2016)

[5] Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., Bischl, B.: OpenML: An R package to connect to the machine learning platform OpenML. Comput. Stat. (2017). https://doi.org/10.1007/s00180-017-0742-2
[6] Cohen, S., Dror, G., Ruppin, E.: Feature selection via coalitional game theory. Neural Comput. **19**(7), 1939–1961 (2007)
[7] Cutler, A., Cutler, D.R., Stevens, J.R.: Random forests. In: Ensemble Machine Learning, pp. 157–175. Springer (2012)
[8] Datta, A., Sen, S., Zick, Y.: Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016 pp. 598–617 (2016)
[9] Fisher, A., Rudin, C., Dominici, F.: Model class reliance: Variable importance measures for any machine learning model class, from the "Rashomon" perspective. arXiv preprint arXiv:1801.01489 (2018)
[10] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
[11] Goldstein, A., Kapelner, A., Bleich, J., Pitkin, E.: Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. J. Comput. Graph. Stat. **24**(1), 44–65 (2015)
[12] Gregorutti, B., Michel, B., Saint-Pierre, P.: Correlation and variable importance in random forests. Stat. Comput. **27**(3), 659–678 (2017)
[13] Lang, M., Bischl, B., Surmann, D.: batchtools: Tools for R to work on batch systems. The Journal of Open Source Software **2**(10) (2017)
[14] Lipton, Z.C.: The mythos of model interpretability. ICML WHI '16 (2016)
[15] Lundberg, S.M., Erion, G.G., Lee, S.I.: Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888 (2018)
[16] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: NIPS, vol. 30, pp. 4765–4774. Curran Associates, Inc. (2017)
[17] Molnar, C., Casalicchio, G., Bischl, B.: iml: An R package for interpretable machine learning. The Journal of Open Source Software **3**(786) (2018)
[18] Serfling, R.J.: Approximation Theorems of Mathematical Statistics, vol. 162. John Wiley & Sons (2009)
[19] Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games **2**(28), 307–317 (1953)
[20] Strobl, C., Boulesteix, A.L., Kneib, T., Augustin, T., Zeileis, A.: Conditional variable importance for random forests. BMC Bioinf. **9**, 307 (2008)
[21] Štrumbelj, E., Kononenko, I., Wrobel, S.: An efficient explanation of individual classifications using game theory. J. Mach. Learn. Res. **11**(Jan), 1–18 (2010)
[22] Štrumbelj, E., Kononenko, I.: A general method for visualizing and explaining black-box regression models. In: Int. Conf. on Adaptive and Natural Computing Algorithms. pp. 21–30. Springer (2011)
[23] Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. ACM SIGKDD Explor. Newsl. **15**(2), 49–60 (2014)

# Part IV.

# Conclusion

# 8. Concluding Remarks and Future Work

The mlr package wraps machine learning algorithms from already existing packages and puts them into a unified input and output structure. It also implements algorithms to handle even more complex supervised learning tasks, such as multilabel classification (see Chapter 3). Although currently not implemented, further extensions to support ordinal regression and mixed-type multi-outcome prediction are planned. There are many methods that transform ordinal regression tasks into multiple binary classification tasks and thus also take into account the ordering imposed by the ordinal target value (Li and Lin, 2006; Gutierrez et al., 2016). Implementing such methods into the mlr package is very similar to the work in Chapter 3. However, these methods involve fitting multiple binary classifiers, which can be computationally demanding. Another possibility to support ordinal regression tasks in mlr is to consider the ordinal target as a real-valued target so that traditional regression algorithms can be used. However, obtaining ordinal classes requires an additional step, namely the search of reasonable thresholds that introduce ordinal categories to the real-valued predictions. Methods that are based on this idea include threshold methods that assume an underlying latent variable (Sánchez-Monedero et al., 2013) and naive methods based on rounding the real-valued predictions (Kramer et al., 2001). An alternative, which requires further investigation, would be to treat the thresholds as hyperparameters and to estimate them by directly optimizing the performance measure of interest. A similar idea was described by the author of this dissertation in a Kaggle competition, which received considerable attention.[1]

Multilabel classification can be seen as a special case of a more general task, i.e., the task of simultaneously modeling and predicting multiple targets of mixed-type. In Saha et al. (2017), a framework for such mixed-type multi-outcome prediction was described, where each target can either be real-valued, discrete or ordinal. Adding support for such tasks in mlr should again be straightforward to the work in Chapter 3. However, assessing the performance in such situations is only straightforward if all targets have the same type, such as only real-valued, only ordinal, or only discrete (either with two or with multiple categories) targets. If the targets are of mixed-type, one of the biggest challenges may be to find an appropriate way on how to assess the performance, which requires future work, especially since the literature on this topic is rather scarce.

One of the aims of OpenML is to make benchmark results and meta-information for algorithms and datasets available to researchers all over the world on a large scale. The hope is that many questions may be answered based on such results. Researchers already use benchmark results in OpenML to answer research questions such as identifying which hyperparameters are most important in several machine learning algorithms and finding generally good hyperparameter values that might be used as defaults in software tools (Probst et al., 2018; van Rijn and Hutter, 2018). Another example is the work of Bilalli et al. (2017), who investigated which characteristics of datasets are responsible for algorithms to perform better. Other similar research questions that might be answered with already existing benchmark results on OpenML are conceivable.

[1] Kaggle. `https://www.kaggle.com/casalicchio/use-the-mlr-package-scores-0-649#102820`. Accessed on November 24th, 2018

In Chapter 6, the RBP curve was proposed to assess the performance of probabilistic classifiers. The RBP curve is based on visualizing the empirical cumulative distribution of the model residuals. An extension to multiclass classification algorithms may also be possible in a one-versus-rest or one-versus-one fashion. Furthermore, as the RBP curve is based on the model residuals and the integral enclosed by the RBP curve is directly related to the $MAE$, it could also be used in the regression setting. In the context of comparing model-agnostic models, a very similar visual tool called *residual distribution plot* was recently implemented in Biecek (2018) and Gosiewska and Biecek (2018). This tool is based on the empirical cumulative distribution function for the *absolute values* of the model residuals. Here, further work will be required to study the role of the RBP curve in the regression setting, its usefulness (e.g., in the comparison and diagnosis of model-agnostic models), and its relation to other visual tools, such as residual plots, which are also based on residuals and are often used for model diagnosis in regression models.

As mentioned, there is a trade-off between model interpretability and model performance. A global measure that takes into account the model interpretability and the model performance may be useful in model selection. This would allow practitioners to visualize the trade-off so that they can choose a Pareto-optimal[2] model regarding the two criteria, namely the model interpretability and the model performance. However, this purpose requires being able to quantify the interpretability of a model. While the model performance can be quantified using specific performance measures, the issue of how practitioners may quantify the interpretability of a model is not a trivial matter and thus requires further research.

The scientific contribution in Chapter 7 was threefold. First, the contribution uses a unified notation based on the partial dependence function, and it gives an encompassing overview of some interpretability methods for feature effect estimation. Furthermore, several techniques for the visualization of feature effects were adapted so that they can be applied to visualize the performance-based feature importance. The work was based on a recently proposed global model-agnostic feature importance, which also served as a basis for the proposed local feature importance measure. Further work may be done using the proposed local feature importance, such as finding or clustering observations with similar local feature importance values. Lastly, the contribution describes another interpretability method for a fair attribution of the feature importance to the features. The variance-based feature importance methods described in the ontology in Section 2.5.2 are based on using the variance of the model predictions. The development of similar methods that use the variance of the model performance based on the work in Chapter 7 is conceivable and needs to be further investigated.

[2]For Pareto-optimal solutions, it is not possible to improve one criterion without worsening another at the same time.

# Contributing Publications

Probst, P., Au, Q., Casalicchio, G., Stachl, C., and Bischl, B. (2017). Multilabel Classification with R Package mlr. *R Journal*, 9(1). *https://journal.r-project.org/archive/2017/RJ-2017-012/index.html*.

Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., and Bischl, B. (2017). OpenML: An R Package to Connect to the Machine Learning Platform OpenML. *Computational Statistics*, pages 1–15. *https://doi.org/10.1007/s00180-017-0742-2*.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2017). OpenML Benchmarking Suites and the OpenML100. *arXiv preprint arXiv:1708.03731*. *https://arxiv.org/abs/1708.03731*. In Preparation.

Casalicchio, G., Bischl, B., Boulesteix, A.-L., and Schmid, M. (2016). The Residual-based Predictiveness Curve: A Visual Tool to Assess the Performance of Prediction Models. *Biometrics*, 72(2):392–401. *https://doi.org/10.1111/biom.12455*.

Casalicchio, G., Molnar, C., and Bischl, B. (2018). Visualizing the Feature Importance for Black Box Models. *arXiv preprint arXiv:1804.06620*. *https://arxiv.org/abs/1804.06620*. To Appear in: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018*.

# Further References


Ancona, M., E. Ceolini, C. Oztireli, and M. Gross (2018). Towards Better Understanding of Gradient-based Attribution Methods for Deep Neural Networks. In *6th International Conference on Learning Representations (ICLR 2018)*.

Androutsopoulos, I., G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos (2000). Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-based Approach. *arXiv preprint cs/0009009*.

Bengio, Y. and Y. Grandvalet (2004). No Unbiased Estimator of the Variance of K-fold Cross-validation. *Journal of Machine Learning Research 5*(Sep), 1089–1105.

Berk, R., L. Sherman, G. Barnes, E. Kurtz, and L. Ahlman (2009). Forecasting Murder within a Population of Probationers and Parolees: A High Stakes Application of Statistical Learning. *Journal of the Royal Statistical Society: Series A (Statistics in Society) 172*(1), 191–211.

Biecek, P. (2018). DALEX: Explainers for Complex Predictive Models in R. *Journal of Machine Learning Research 19*(84), 1–5.

Bilalli, B., A. Abelló, and T. Aluja-Banet (2017). On the Predictive Power of Meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science 27*(4), 697–712.

Biran, O. and C. Cotton (2017). Explanation and Justification in Machine Learning: A Survey. In *IJCAI-17 Workshop on Explainable AI (XAI)*, pp. 8.

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

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag.

Bleik, S. (2015). Permutation Feature Importance – Machine Learning Blog. Microsoft Corporation. URL `https://blogs.technet.microsoft.com/machinelearning/2015/04/14/permutation-feature-importance/`. Accessed: 2018-11-23.

Boulesteix, A.-L., R. Hable, S. Lauer, and M. J. Eugster (2015). A Statistical Framework for Hypothesis Testing in Real Data Comparison Studies. *The American Statistician 69*(3), 201–212.

Box, G. (1979). Robustness in the Strategy of Scientific Model Building. In R. L. Launer and G. N. Wilkinson (Eds.), *Robustness in Statistics*, pp. 201–236. New York: Academic Press.

Bradley, R. A. and M. E. Terry (1952). Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika 39*(3/4), 324–345.

Braun, M. L. and C. S. Ong (2014). Open Science in Machine Learning. In V. Stodden, F. Leisch, and R. D. Peng (Eds.), *Implementing Reproducible Research*. Chapman and Hall/CRC.

Breiman, L. (2001). Random Forests. *Machine Learning 45*(1), 5–32.

Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. *Monthey Weather Review 78*(1), 1–3.

Brodersen, K. H., C. S. Ong, K. E. Stephan, and J. M. Buhmann (2010). The Balanced Accuracy and its Posterior Distribution. In *Proceedings of the 2010 20$^{th}$ International Conference on Pattern Recognition*, pp. 3121–3124. IEEE.

Buja, A., W. Stuetzle, and Y. Shen (2005). Loss Functions for Binary Class Probability Estimation and Classification: Structure and Applications. *Working Paper*. URL `http://www-stat.wharton.upenn.edu/~buja/PAPERS/paper-proper-scoring.pdf`.

Caruana, R., Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad (2015). Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission. *Proceedings of the 21$^{th}$ ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, 1721–1730.

Casalicchio, G., G. Tutz, and G. Schauberger (2015). Subject–specific Bradley–Terry–Luce Models with Implicit Variable Selection. *Statistical Modelling 15*(6), 526–547.

Charte, F. and D. Charte (2015). Working with Multilabel Datasets in R: The mldr Package. *R Journal 7*(2).

Chen, Y., E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista (2015). The UCR Time Series Classification Archive. URL `http://www.cs.ucr.edu/~eamonn/time_series_data/`.

Cook, N. R. (2007). Use and Misuse of the Receiver Operating Characteristic Curve in Risk Prediction. *Circulation 115*(7), 928–935.

Cramer, H. (1946). *Mathematical Methods of Statistics*. Princeton, NJ, USA: Princeton University Press.

Cutler, D. R., T. C. Edwards, K. H. Beard, A. Cutler, K. T. Hess, J. Gibson, and J. J. Lawler (2007). Random Forests for Classification in Ecology. *Ecology 88*(11), 2783–2792.

Dheeru, D. and E. Karra Taniskidou (2017). UCI Machine Learning Repository. URL `http://archive.ics.uci.edu/ml`.

Domingos, P. (2015). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York, NY, USA: Basic Books.

Doshi-Velez, F. and B. Kim (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*.

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics 7*(1), 1–26.

Efron, B. (1983). Estimating the Error Rate of a Prediction Rule: Improvement on Cross-validation. *Journal of the American Statistical Association 78*(382), 316–331.

Eugster, M. J., T. Hothorn, and F. Leisch (2012). Domain-based Benchmark Experiments: Exploratory and Inferential Analysis. *Austrian Journal of Statistics 41*(1), 5–26.

Eugster, M. J., F. Leisch, and C. Strobl (2014). (Psycho-)Analysis of Benchmark Experiments: A Formal Framework for Investigating the Relationship between Data Sets and Learning Algorithms. *Computational Statistics & Data Analysis 71*, 986–1000.

Fawcett, T. (2004). ROC Graphs: Notes and Practical Considerations for Researchers. *Machine Learning 31*(1), 1–38.

Fernández-Delgado, M., E. Cernadas, S. Barro, and D. Amorim (2014). Do We Need Hundreds of Classifiers to Solve Real World Classification Problems. *Journal of Machine Learning Research 15*(1), 3133–3181.

Ferri, C., J. Hernández-Orallo, and R. Modroiu (2009). An Experimental Comparison of Performance Measures for Classification. *Pattern Recognition Letters 30*(1), 27–38.

Fisher, A., C. Rudin, and F. Dominici (2018). Model Class Reliance: Variable Importance Measures for Any Machine Learning Model Class, from the "Rashomon" Perspective. *arXiv preprint arXiv:1801.01489*.

Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. New York, NY, USA: Cambridge University Press.

Fluss, R., D. Faraggi, and B. Reiser (2005). Estimation of the Youden Index and its Associated Cutoff Point. *Biometrical Journal 47*(4), 458–472.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics 29*(5), 1189–1232.

Gilpin, L. H., D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal (2018). Explaining Explanations: An Approach to Evaluating Interpretability of Machine Learning. *arXiv preprint arXiv:1806.00069*.

Giraud-Carrier, C. (2008). Metalearning – A Tutorial. In *Tutorial at the 2008 International Conference on Machine Learning and Applications, ICMLA*.

Giraud-Carrier, C. and F. Provost (2005). 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*, pp. 12–19.

Gneiting, T. and A. E. Raftery (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association 102*(477), 359–378.

Goldstein, A., A. Kapelner, J. Bleich, and E. Pitkin (2015). Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics 24*(1), 44–65.

Gosiewska, A. and P. Biecek (2018). auditor: An R Package for Model-Agnostic Visual Validation and Diagnostic. *arXiv preprint arXiv:1809.07763*.

Greenwell, B. M., B. C. Boehmke, and A. J. McCarthy (2018). A Simple and Effective Model-based Variable Importance Measure. *arXiv preprint arXiv:1805.04755*.

Guidotti, R., A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi (2018). A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys (CSUR) 51*(5), 93.

Gutierrez, P. A., M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez (2016). Ordinal Regression Methods: Survey and Experimental Study. *IEEE Transactions on Knowledge and Data Engineering 28*(1), 127–146.

Hand, D. J. and R. J. Till (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning 45*(2), 171–186.

Hanley, J. A. and B. J. McNeil (1982). The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology 143*(1), 29–36.

Hernández-Orallo, J., P. Flach, and C. Ferri (2012). A Unified View of Performance Metrics: Translating Threshold Choice into Expected Classification Loss. *Journal of Machine Learning Research 13*(Oct), 2813–2869.

Hernández-Orallo, J., P. A. Flach, and C. F. Ramirez (2011). Brier Curves: A New Cost-based Visualisation of Classifier Performance. In *Proceedings of the 28$^{th}$ International Conference on International Conference on Machine Learning*, pp. 585–592.

Hidalgo, J. M. G. (2002). Evaluating Cost-sensitive Unsolicited Bulk Email Categorization. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 615–620. ACM.

Hill, D. W. and Z. M. Jones (2014). An Empirical Evaluation of Explanations for State Repression. *American Political Science Review 108*(3), 661–687.

Hofner, B., M. Schmid, and L. Edler (2016). Reproducible Research in Statistics: A Review and Guidelines for the Biometrical Journal. *Biometrical Journal 58*(2), 416–427.

Holmberg, L. and A. Vickers (2013). Evaluation of Prediction Models for Decision-Making: Beyond Calibration and Discrimination. *PLoS Medicine 10*(7), 1–2.

Hooker, G. (2004). Discovering Additive Structure in Black Box Functions. In *Proceedings of the 10$^{th}$ ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 575–580. ACM.

Hooker, G. (2007). Generalized Functional Anova Diagnostics for High-dimensional Functions of Dependent Variables. *Journal of Computational and Graphical Statistics 16*(3), 709–732.

Hosmer, D. W. and S. Lemesbow (1980). Goodness of Fit Tests for the Multiple Logistic Regression Model. *Communications in Statistics – Theory and Methods 9*(10), 1043–1069.

Hothorn, T. (2018). Cran Task View: Machine Learning & Statistical Learning. Version 2018-08-05, URL `https://CRAN.R-project.org/view=MachineLearning`.

Hothorn, T. and F. Leisch (2011). Case Studies in Reproducibility. *Briefings in Bioinformatics 12*(3), 288–300.

Hothorn, T., F. Leisch, A. Zeileis, and K. Hornik (2005). The Design and Analysis of Benchmark Experiments. *Journal of Computational and Graphical Statistics 14*(3), 675–699.

Hripcsak, G. and A. S. Rothschild (2005). Agreement, the F-Measure, and Reliability in Information Retrieval. *Journal of the American Medical Informatics Association 12*(3), 296–298.

Huang, Y., M. S. Pepe, and Z. Feng (2007). Evaluating the Predictiveness of a Continuous Marker. *Biometrics 63*(4), 1181–1188.

Iooss, B. and P. Lemaître (2015). A Review on Global Sensitivity Analysis Methods. In *Uncertainty Management in Simulation-Optimization of Complex Systems*, pp. 101–122. Boston: Springer.

Japkowicz, N. and M. Shah (2011). *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge: Cambridge University Press.

Jiang, T. and A. B. Owen (2002). Quasi-regression for Visualization and Interpretation of Black Box Functions. Technical report. URL `http://statweb.stanford.edu/~owen/reports/qregvis.pdf`.

Kim, J.-H. (2009). Estimating Classification Error Rate: Repeated Cross-validation, Repeated Hold-out and Bootstrap. *Computational Statistics & Data Analysis 53*(11), 3735–3745.

Korobov, M. and K. Lopuhin (2017). Permutation Importance – ELI5 0.8.1 Documentation. URL `https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html`. Accessed: 2018-11-23.

Kramer, S., G. Widmer, B. Pfahringer, and M. De Groeve (2001). Prediction of Ordinal Classes Using Regression Trees. *Fundamenta Informaticae 47*(1-2), 1–13.

Krause, J., A. Perer, and E. Bertini (2016). Using Visual Analytics to Interpret Predictive Machine Learning Models. *arXiv preprint arXiv:1606.05685*.

Leeper, T. J. (2017). Interpreting Regression Results Using Average Marginal Effects with R's Margins. Technical report. URL `https://cran.r-project.org/web/packages/margins/index.html`.

Leisch, F. (2002). Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis. In *Compstat*, pp. 575–580. Springer.

Lemeshow, S. and D. W. Hosmer (1982). A Review of Goodness of Fit Statistics for Use in the Development of Logistic Regression Models. *American Journal of Epidemiology 115*(1), 92–106.

Li, L. and H.-T. Lin (2006). Ordinal Regression by Extended Binary Classification. In *Proceedings of the 19$^{th}$ International Conference on Neural Information Processing Systems*, pp. 865–872. MIT Press.

Lipton, Z. C. (2016). The Mythos of Model Interpretability. *arXiv preprint arXiv:1606.03490*.

Lucena, B. (2018). Spline-Based Probability Calibration. *arXiv preprint arXiv:1809.07751*.

Lundberg, S. and S.-I. Lee (2017). A Unified Approach to Interpreting Model Predictions. *arXiv preprint arXiv:1705.07874*.

Lundberg, S. M., G. G. Erion, and S.-I. Lee (2018). Consistent Individualized Feature Attribution for Tree Ensembles. *arXiv preprint arXiv:1802.03888*.

Miller, T. (2017). Explanation in Artificial Intelligence: Insights from the Social Sciences. *arXiv preprint arXiv:1706.07269*.

Molinaro, A. M., R. Simon, and R. M. Pfeiffer (2005). Prediction Error Estimation: A Comparison of Resampling Methods. *Bioinformatics 21*(15), 3301–3307.

Molnar, C., G. Casalicchio, and B. Bischl (2018). iml: An R package for Interpretable Machine Learning. *The Journal of Open Source Software 3*(26), 786.

Murphy, A. H. (1972). Scalar and Vector Partitions of the Probability Score: Part I. Two-State Situation. *Journal of Applied Meteorology 11*(2), 273–282.

Murphy, A. H. (1973). A New Vector Partition of the Probability Score. *Journal of Applied Meteorology 12*(4), 595–600.

Murphy, K. P. (2013). *Machine Learning: A Probabilistic Perspective* (1 ed.). MIT Press.

Niculescu-Mizil, A. and R. Caruana (2005). Predicting Good Probabilities with Supervised Learning. *Proceedings of the 22$^{nd}$ International Conference on Machine Learning - ICML '05* (1999), 625–632.

Obermeyer, Z. and E. J. Emanuel (2016). Predicting the Future – Big Data, Machine Learning, and Clinical Medicine. *The New England Journal of Medicine 375*(13), 1216.

Olson, R. S., W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore (2017). PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *BioData Mining 10*(1), 36.

Owen, A. B. (2014). Sobol'indices and Shapley Value. *SIAM/ASA Journal on Uncertainty Quantification 2*(1), 245–251.

Pencina, M. J., R. B. D'Agostino Sr, R. B. D'Agostino Jr, and R. S. Vasan (2008). Evaluating the Added Predictive Ability of a New Marker: From Area under the ROC Curve to Reclassification and Beyond. *Statistics in Medicine 27*(2), 157–172.

Pepe, M. S., Z. Feng, Y. Huang, G. Longton, R. Prentice, I. M. Thompson, et al. (2008, February). Integrating the Predictiveness of a Marker with its Performance as a Classifier. *American Journal of Epidemiology 167*(3), 362–368.

Perkins, N. J. and E. F. Schisterman (2006). The Inconsistency of "Optimal" Cutpoints Obtained Using Two Criteria Based on the Receiver Operating Characteristic Curve. *American Journal of Epidemiology 163*(7), 670–675.

Platt, J. et al. (1999). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers 10*(3), 61–74.

Probst, P., B. Bischl, and A.-L. Boulesteix (2018). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *arXiv preprint arXiv:1802.09596*.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Ribeiro, M. T., S. Singh, and C. Guestrin (2016). Model-Agnostic Interpretability of Machine Learning. *arXiv preprint arXiv:1606.05386*.

Saha, B., S. Gupta, D. Phung, and S. Venkatesh (2017). A Framework for Mixed-Type Multi-outcome Prediction With Applications in Healthcare. *IEEE Journal of Biomedical and Health Informatics 21*(4), 1182–1191.

Sánchez-Monedero, J., P. A. Gutiérrez, P. Tiňo, and C. Hervás-Martínez (2013). Exploitation of Pairwise Class Distances for Ordinal Classification. *Neural Computation 25*(9), 2450–2485.

Schaffer, C. (1994). A Conservation Law for Generalization Performance. In *Machine Learning Proceedings 1994*, pp. 259–265. Elsevier.

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games 2*(28), 307–317.

Shynk, J. J. (2012). *Probability, Random Variables, and Random Processes: Theory and Signal Processing Applications*. John Wiley & Sons.

Sobol', I. M. (1990). On Sensitivity Estimation for Nonlinear Mathematical Models. *Matematicheskoe Modelirovanie 2*(1), 112–118.

Sokolova, M., N. Japkowicz, and S. Szpakowicz (2006). Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. In *Australasian Joint Conference on Artificial Intelligence*, pp. 1015–1021. Springer.

Sokolova, M. and G. Lapalme (2009). A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing & Management 45*(4), 427–437.

Steyerberg, E., A. Vickers, N. Cook, T. Gerds, M. Gonen, N. Obuchowski, et al. (2010). Assessing the Performance of Prediction Models. *Epidemiology 21*(1), 128–138.

Stodden, V., F. Leisch, and R. D. Peng (2014). *Implementing Reproducible Research*. CRC Press.

Strumbelj, E. and I. Kononenko (2014). Explaining Prediction Models and Individual Predictions with Feature Contributions. *Knowledge and Information Systems 41*(3), 647–665.

Team, A. (2016). AzureML: Anatomy of a Machine Learning Service. In *Conference on Predictive APIs and Apps*, pp. 1–13.

van Rijn, J. N. and F. Hutter (2018). Hyperparameter Importance across Datasets. In *Proceedings of the 24$^{th}$ ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2367–2376. ACM.

van Someren, M. (2001). Model Class Selection and Construction: Beyond the Procrustean Approach to Machine Learning Applications. *Machine Learning and Its Applications*, 196–217.

VanderPlas, J., A. J. Connolly, Ž. Ivezić, and A. Gray (2012). Introduction to astroML: Machine Learning for Astrophysics. In *2012 Conference on Intelligent Data Understanding*, pp. 47–54. IEEE.

Vanschoren, J., J. N. van Rijn, B. Bischl, and L. Torgo (2013). OpenML: Networked Science in Machine Learning. *SIGKDD Explorations 15*(2), 49–60.

Vilalta, R. and Y. Drissi (2002). A Perspective View and Survey of Meta-learning. *Artificial Intelligence Review 18*(2), 77–95.

Wang, T., C. Rudin, D. Wagner, and R. Sevieri (2013). Learning to Detect Patterns of Crime. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 515–530. Springer.

Wei, P., Z. Lu, and J. Song (2015). Variable Importance Analysis: A Comprehensive Review. *Reliability Engineering & System Safety 142*, 399–432.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. New York, NY, USA: Springer.

Wickham, H. (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1, URL `https://CRAN.R-project.org/package=tidyverse`.

Wolpert, D. H. (1996). The Lack of a Priori Distinctions between Learning Algorithms. *Neural Computation 8*(7), 1341–1390.

Xie, Y. (2014). knitr: A Comprehensive Tool for Reproducible Research in R. In V. Stodden, F. Leisch, and R. D. Peng (Eds.), *Implementing Reproducible Research*. Chapman and Hall/CRC.

Zadrozny, B. and C. Elkan (2002). Transforming Classifier Scores into Accurate Multiclass Probability Estimates. In *Proceedings of the $8^{th}$ ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 694–699. ACM.

Zhang, M.-L. and Z.-H. Zhou (2014). A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering 26*(8), 1819–1837.

Zhao, Q. and T. Hastie (2017). Causal Interpretations of Black-box Models. *Journal of Business & Economic Statistics, to appear*.

Zien, A., N. Krämer, S. Sonnenburg, and G. Rätsch (2009). The Feature Importance Ranking Measure. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 694–709. Springer.

# Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12. Juli 2011, § 8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 07.01.2019 Giuseppe Casalicchio